Tapaustutkimus ketterän kehittämisen toimintatapojen eroista finanssialan yrityksessä

Pauli Kostamo

Pro gradu -tutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen osasto

Helsinki, 6. huhtikuuta 2020

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Matemaattis-luonnontieteellinen	Tietojenkäsittelytieteen osasto

Tekijä — Författare — Author
Pauli Kostamo

Työn nimi — Arbetets titel — Title

Tapaustutkimus ketterän kehittämisen toimintatapojen eroista finanssialan yrityksessä

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Pro gradu -tutkielma	6. huhtikuuta 2020	63

Tiivistelmä — Referat — Abstract

Ketterät menetelmät ja Lean-menetelmät ovat nykyaikaisessa ohjelmistokehityksessä useiden käyttäjätutkimusten perusteella käytetyimpiä tapoja kehittää ohjelmistoja. Ketteriä ja Lean menetelmiä on kehitetty jo pitkään, mutta niiden suosion kasvu on ollut suurinta 2000-luvun alun jälkeen. Alun perin ketterät ja Lean -menetelmät olivat pienille paikallisille tiimeille suunniteltuja. Niiden osoittauduttua perinteisiä menetelmiä toimivimmiksi, niistä on pyritty luomaan malleja myös suurille hajautetuille organisaatioille, joissa useat tiimit toimivat synkronoidusti.

Tässä tutkielmassa selvitetään puolistrukturoiduista teemahaastatteluista tehdyn kvalitatiivisen analyysin avulla kahden samassa korporaatiossa toimivan ohjelmistokehitystiimin toimintaa. Tiimien rakenne on keskenään erilainen. Toinen tiimeistä käyttää Scaled Agile Framework (SAFe) -ohjelmistokehityskehystä korporaation pääbrändin alaisuudessa yhdessä useamman tiimin kanssa. Toinen tiimi taas työskentelee oman brändinsä alaisuudessa käyttäen hyvin vapaata Scrum/ Kanban tyylistä ohjelmistokehitysmenetelmää.

Tutkielmassa selvitetään yhtäläisyyksiä ja eroja näiden tiimien toimintatavoista, sekä selvitetään, onko tiimeillä käsite-eroja ketterästä kehittämisestä. Tutkielmassa selvitetään myös kuinka tiimit kokevat hajautetun tiimin kanssa toiminnan vaikuttavan työntekoonsa. Tutkimuksessa pyritään myös löytämään mahdollisia syitä suurille yrityksille suunnatun SAFe:n ketterän kehityksen mallin toteuttamisen haasteisiin vertaamalla muiden tutkimusten tuloksia haastatellun SAFe-tiimin kokemuksiin. Tutkimustulosten pohjalta pyritään löytämäänkeinoja kehittää ja tehostaa yrityksen ohjelmistokehityskulttuuria.

Tutkimuksessa ilmenee tiimien organisaatiorakenteen eroavan huomattavasti toisistaan. Tiimeillä havaitaan olevan käsite-eroja ketterän kehittämisen periaatteesta, jossa muuttuviin vaatimuksiin reagoidaan nopeasti. SAFe-tiimi on havainnut organisaatiossaan olevan muutosvastarintaa, eivätkä kaikki osastot ole sitoutuneet käyttämään SAFe-mallia. Sama havainto on tehty useissa muissakin tutkimuksissa. Scrum-tiimin mukaan kahden viikon sprinttisykli on liian pitkä ollakseen ketterää kehittämistä. Analyysissä havaitaan hajautetun tiimin toiminnan vaativan erityisjärjestelyitä toimiakseen mahdollisimman tehokkaasti.

ACM Computing Classification System (CCS):
Software and its engineering
Software creation and management
Software development process management
Software development methods
Agile software development

Avainsanat — Nyckelord — Keywords
SAFe, Scrum, Kanban, Agile, Lean, Ketterä kehittäminen

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Sisältö

1 Johdanto

Käytän tutkimuksessani *kursivoitua* tekstiä esitellessäni uuden termin.

Ketterät ohjelmistokehitysmenetelmät ovat kasvattaneet suosiotaan 2000-luvun alun jälkeen. Erilaisia ketteriä menetelmiä on kehitetty erikokoisille organisaatioille ja toimintatavoille useita. Lukuisten epätieteellisten raporttien mukaan ketterän kehityksen menetelmät ovat ohjelmistokehityksessä joko yhtä yleisiä tai jopa yleisempiä kuin perinteiset ohjelmistokehitysmenetelmät.

Ohjelmistokehitysmenetelmät voidaan jakaa karkeasti kahteen pääluokkaan: perinteisiin ohjelmistokehitysmenetelmiin ja ketteriin/ *Lean* -menetelmiin [NB07]. Perinteisiin ohjelmistokehitysmenetelmiin luetaan vaiheittain vain yhteen suuntaan etenevät ohjelmistokehitysmenetelmät kuten vesiputousmalli, johon perehdytään tarkemmin luvussa 2. Ketterät ohjelmistokehitysmenetelmät ja Lean-menetelmät koostuvat syklisestä ohjelmistokehityksestä, jossa ohjelmistoa kehitetään jatkuvasti iteroiden [NB07]. Lean ja ketterä kehitys eivät ole täysin sama asia, mutta molemmat menetelmät pohjautuvat pitkälti samaan filosofiaan ja termejä käytetään yleisesti vastaamaan toisiaan [Wan11]. Ohjelmistokehityksessä Lean-menetelmistä tällä hetkellä tunnetuin on *Kanban* ja ketteristä menetelmistä eräät tunnetuimmat menetelmät ovat *Scrum* ja *Extreme Programming (XP)*. Luvussa 2.1 tutustutaan tarkemmin Scrum- ja Kanban-malleihin.

Ketterän kehityksen mallit on alun perin suunniteltu pienille paikallisille tiimeille, mutta niiden osoittauduttua toimivimmiksi kuin perinteisten ohjelmistokehitysmenetelmien, ketterän kehityksen prosessien skaalaaminen suuremmillekin organisaatioille on tullut ajankohtaiseksi. Tällä hetkellä suurissa organisaatioissa ketterän kehittämisen menetelmien käyttöön ottaminen on ajankohtainen aihe. Suurille organisaatioille on kehitetty useita ketterän kehityksen ohjelmistokehityskehyksiä, kuten *Scaled Agile Framework (SAFe)*, *Large Scale Scrum (LeSS)* ja *Disciplined Agile Delivery (DAD)*. Näistä SAFe on tällä hetkellä suosituin skaalatun ketterän kehittämisen ohjelmistokehityskehys [PPL18] [Ver].

Sekä Scrum että Kanban ovat matalan tason ketterän kehityksen malleja, joka ottavat ohjelmistokehityksen viitekehyksessä kantaa vain kehitystiimin ja tuoteomistajan rooleihin ja toimintatapoihin. Ne tarjoavat menetelmiä yksinkertaiselle tuotekehitysiteraatioille, mutta eivät ota kantaa hallinnollisiin tehtäviin. Scaled Agile Framework (SAFe) taas on korkean tason ketterän kehityksen kehys, joka tarjoaa työskentelymenetelmiä organisaation laajuisesti. SAFe määrittelee roolit, vastuualueet ja toimintamallit ketterään ohjelmistokehitykseen. Kehyksen periaate on tehdä ketterästä ohjelmistokehityksestä skaalautuvampaa ja etenkin suuremmille organisaatioille toimivampaa mallia. SAFe-mallin ideana on auttaa ketterän kehityksen metodologioiden levittämistä organisaation laajuisesti, auttaa tekemään kehitystyötä alan kärkitekniikoilla ja tarjota metodeja yritysarvon kasvattamiseen [Col15, s. 114]. Vaikka SAFe on suosituin suurille yrityksille suunnattu ketterän kehityksen ohjelmistokehityskehys, sen hyödyistä ja haasteista on julkaistu vain vähän tieteellisiä tutkimuksia [PPL18].

Tässä työssä tutkitaan kahta vaatimusmäärittelyiltään identtistä projektia, jotka on toteutettu samaan aikaan suuressa suomalaisessa finanssialan yrityksessä. Molemmissa projekteissa toteutetut ohjelmat täyttävät Finanssivalvonnan asettamat tarkat vaatimukset henkilön vahvan tunnistamisen toteuttamisesta sekä henkilötietojen käsittelystä. Tutkimus tehtiin kvalitatiivisena tutkimuksena haastatellen projekteissa mukana olevia työntekijöitä puolistrukturoidun haastattelun menetelmin. Toinen projekti on toteutettu Scaled Agile Framework (SAFe) -ohjelmistokehityskehyksellä, ja toinen on toteutettu hyvin vapaana Scrum-/Kanban -tyyppisenä projektina.

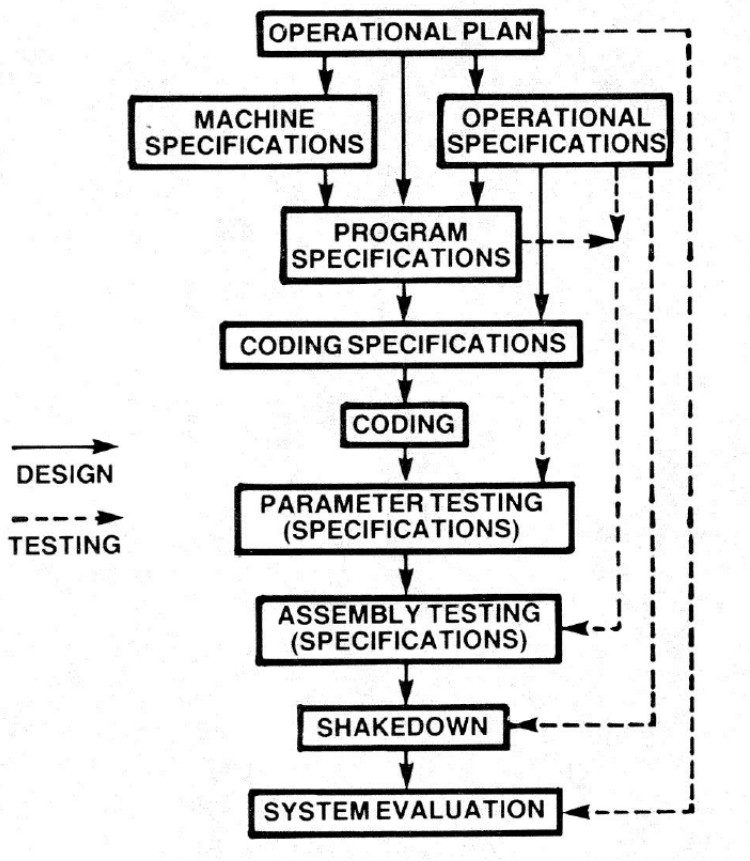
Tässä tutkimuksessa SAFe:n tarkkaan määritellystä organisaatiokaaviosta sekä vapaasta Scrum-/Kanban-tyyppisestä ohjelmistokehityksestä etsitään yhtäläisyyksiä ja eroja. Näiden kautta selvitetään, onko tiimeillä käsite-eroja ketterästä kehittämisestä ja kuinka tiimit kokevat hajautetun tiimin kanssa toiminnan vaikuttavan työntekoon. Tämän lisäksi työssä pyritään löytämään mahdollisia syitä suurille yrityksille suunnatun SAFe:n ketterän kehityksen mallin

toteuttamisen haasteisiin vertaamalla muiden tutkimusten tuloksia haastatellun SAFe-tiimin kokemuksiin. Tutkimustulosten pohjalta pyritään löytämään keinoja kehittää ja tehostaa yrityksen ohjelmistokehityskulttuuria.

2 Ketterät ja Lean ohjelmistokehitysmenetelmät

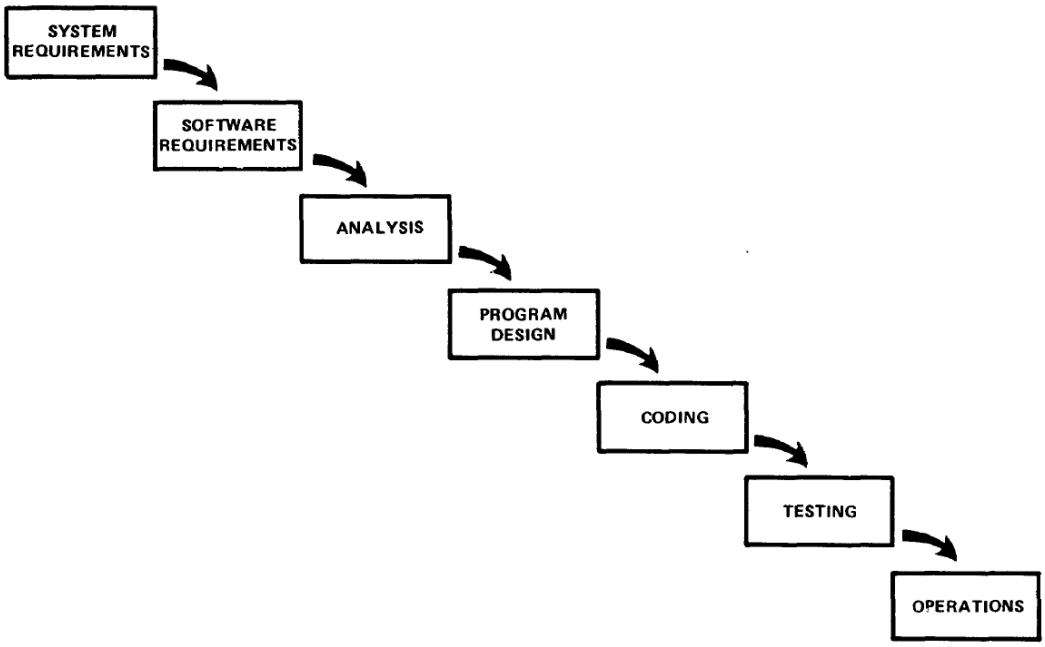
Ketterät ohjelmistokehitysmenetelmät (agile methods) ovat nykyään hyvin suosittuja. Niiden suosio on kasvanut etenkin vuonna 2001 julkaistun *Ketterän ohjelmistokehityksen julistuksen (Manifesto for Agile Software Development)* jälkeen. Ketterän ohjelmistokehityksen idea ei kuitenkaan itsessään ole näin uusi, vaan perinteisiä ohjelmistokehitysmenetelmiä on kritisoitu jo pitkään ja useat parannusehdotukset ovat olleet juuri ketterän kehityksen menetelmien mukaisia, vaikka niitä ei ketteriksi menetelmiksi olisikaan aiemmin kutsuttu [AGW08].

Stack Overflown 2018 tekemän kyselyn mukaan ammattimaisista sovelluskehittäjistä ketteriä menetelmiä käyttää jopa 85,9% [Ove], Gartnerin 2017 tekemän tutkimuksen mukaan ketteriä menetelmiä, Lean-menetelmiä tai muuten iteratiivisia menetelmiä kehityksessä käyttää 61% kyselyyn vastanneista yrityksistä [MW] ja Project Management Instituten 2018 tekemän kyselyn mukaan 46% kyselyyn vastanneista organisaatioista käyttää ketterää tai ketterään viittaavia menetelmiä [Ins]. CollabNet VersionOne:n 2018 tekemän kyselyn mukaan käytetyin ketterän kehityksen metodi oli Scrum 54% osuudellaan, ja suurille organisaatioille skaalatussa ketterässä kehityksessä käytetyin metodi oli Scaled Agile Framework (SAFe) 30% osuudellaan [Ver]. Shine Technologies teki vuonna 2003 maailmanlaajuisen kyselytutkimuksen ketterästä kehittämisestä. Kyselyn tuloksena oli, että ketterän kehittämisen menetelmiä käyttävillä yrityksillä oli parempi tuottavuus, parempi tuotteiden laatu, parempi asiakastyytyväisyys ja pienemmät kulut kuin yrityksillä, jotka eivät käytä ketteriä menetelmiä [Sub10].



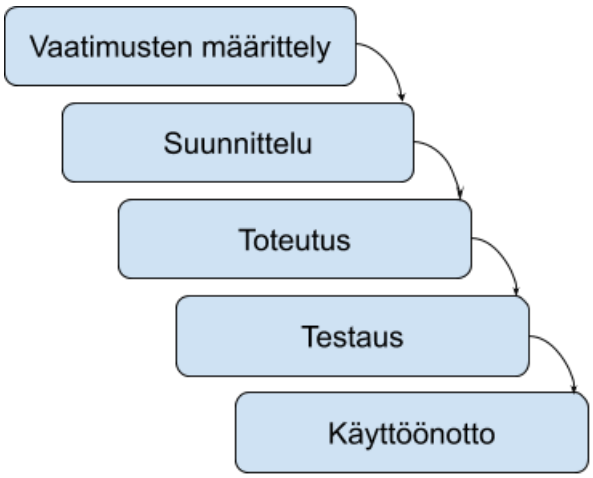
Kuva 1: Lincoln Laboratoryn käyttämä malli suurten ohjelmistosysteemien valmistamiseen [Ben83].

Perinteisesti ohjelmistokehitystä on tehty teollisuudesta tutuin menetelmin, jossa tuote aluksi suunnitellaan huolella ja tämän jälkeen valmistetaan. Yksi ensimmäisistä monimutkaisten tietokoneohjelmien suunnittelumalleista oli 1950-luvun alkupuolella Lincoln Laboratoryn käyttämä portaittainen malli joka on esitetty kuvassa 1. Lincoln Laboratoryn malli on kuitenkin vain ideaalimalli, eikä sen käyttö tällaisenaan toiminut ohjelmistokehityksessä. Mallia käytettäessä etenkin *ohjelmointi* (*Coding*) -portaasta täytyy usein palata takaisin *ohjelmoinnin spesifikaatio* (*Coding Specifications*) -portaaseen, sillä ohjelmoidessa spesifikaatiosta paljastuu usein epäjohdonmukaisuuksia [Ben83]. Portaittaiseen malliin pohjautuvat mallit ovat kuitenkin olleet erittäin käytettyjä ohjelmistokehityksessä.



Kuva 2: Winston Roycen esimerkki toimimattomasta portaittaisesta ohjelmistokehitysmenetelmästä [Roy87]

Portaittaista mallia kutsutaan usein vesiputousmalliksi. Tämän mallin, joskaan ei tällä nimellä, esitteli Winston Royce kritisoidessaan portaittaisen mallin käyttöä vuonna 1987. Eräs Roycen esimerkki toimimattomasta ohjelmistokehitysmenetelmästä on esitettynä kuvassa 2. Tästä mallista on johdettu vesiputousmallina tunnettu ohjelmistokehitysmenetelmä, joka on esitettynä kuvassa 3. Mallia käytetään usein kritisoitaessa jäykäksi koettuja portaittain eteneviä ohjelmistokehitysmenetelmiä. Royce totesi vain yhteen suuntaan etenevässä portaittaisessa mallissa olevan paljon riskejä ja ehdotti osittaisratkaisuksi iteratiivista ohjelmistokehitysmallia [Roy87].



Kuva 3: Vesiputousmalliksi kutsutun ohjelmistokehitysmenetelmän tuotantovaiheet

Portaittain etenevää mallia on kritisoitu jo 1970-luvulta lähtien. Mallia on pidetty ongelmallisena etenkin seuraavista syistä:

- Tuotteen loppukäyttäjän on vaikea tietää tarkasti mitä tuotteelta haluaa.
- Vaikka kaikki vaatimukset tiedostettaisiin, niin kehitystyön edetessä löytyy paljon detaljeja, joita ei voitu tietää ennen toteutusvaihetta.
- Vaikka kaikki kehityksen aikana ilmenneet detaljitkin tiedettäisiin, ihminen ei kykene hallitsemaan suuren systeemin kompleksisuutta.
- Vaikka kompleksisuus pystyttäisiin hallitsemaan, ulkoiset vaatimukset aiheuttavat muutostarpeita ohjelmistoon, joiden takia aiempia päätöksiä täytyy perua.

[LB03]

Ratkaisuna kritiikkiin on pidetty iteratiivista, evoluutionääristä ja inkrementaalista ohjelmistokehitysmallia [LB03], josta ketterä kehitys on saanut alkunsa.

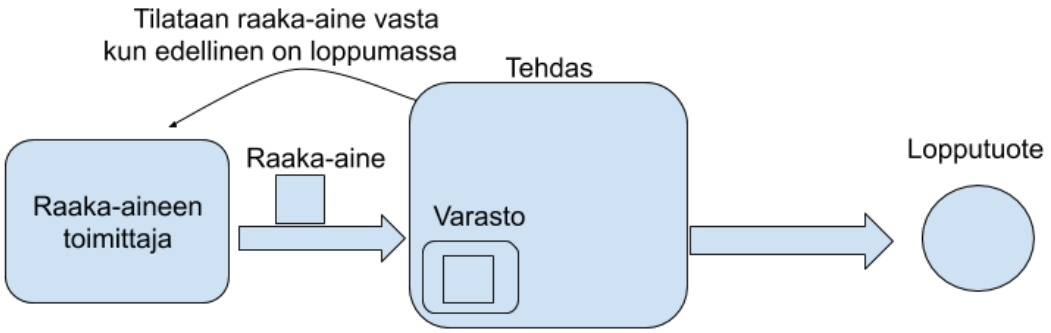
Harlan Mills on todennut muun muassa, että sovelluskehitys kuuluisi tehdä iteratiivisesti jatkuvassa yhteistyössä loppukäyttäjän kanssa, jolloin ohjelmistoa voidaan suunnitella uudelleen koko prosessin ajan [LB03]. Ketterissä menetelmissä ohjelmistoa suunnitellaan ja kehitetään pienissä palasissa. Tällaisissa ohjelmistokehitysmenetelmissä palataan syklisesti aina uudelleen työtehtäviin, jotka vastaavat kuvassa 3 esitetyn vesiputosmallin portaittaisia vaiheita. Ketterien menetelmien tarkoituksena on ollut mahdollistaa nopeampi reagointi ohjelmiston muuttuviin vaatimuksiin. Alun perin ketterät menetelmät on suunniteltu pieniä samassa paikassa työskenteleviä tiimejä varten, mutta nykyään samoja hyväksi havaittuja menetelmiä pyritään hyödyntämään myös suurissa globaalisti hajautetuissa ohjelmistokehitysprojekteissa [PLH12, s. 235].

Useiden tutkimusten mukaan ketterien mallien käyttö ohjelmistokehityksessä on toimivampaa kuin perinteisten ohjelmistokehitysmallien. Ketterien mallien on raportoitu parantaneen ohjelmistokehitystä useilla eri osa-alueilla [DD08]. Myös SAFe-ohjelmistokehityskehyksen käytön on raportoitu parantavan ohjelmistokehitystä useilla eri osa-alueilla. Erään tutkimuksen mukaan SAFe parantaa markkinoille tuloaikaa, nopeuttaa ohjelmiston iteraatioiden toimittamista, parantaa ennustettavuutta, laatua ja tuottavuutta, sekä lisää projektien läpinäkyvyyttä [PPL18].

SAFe:a on kritisoitu muun muassa siitä, että se lisää prosessiin turhaa byrokratiaa. SAFe:a on kutsuttu jopa uudeksi vesiputousmalliksi [EP17, s. 99-100]. Scrum ja Kanban, toisin kuin SAFe, eivät ota lainkaan kantaa organisaation hallinnollisiin rakenteisiin. Eräässä tutkimuksessa on havaittu, että ketterän kehityksen metodien skaalaaminen suuriin järjestelmiin on haastavaa, sillä suuret järjestelmät vaativat paljon etukäteissuunnittelua ja dokumentointia. Tutkimuksen mukaan myös *jatkuva integraatio* (*continuous integration, CI*) on käytännössä jopa mahdotonta, kun samaa järjestelmää kehittää useampi kehitystiimi samaan aikaan [PG14]. Samassa tutkimuksessa esitetään myös väite, että SAFe:n fundamentaalinen olettamus on, että kun organisaatio on iso, ketterän kehityksen mallien täytyy skaalautua isolle organisaatiolle. Tutkimuksen mukaan väite on väärä, ja jopa isoissa organisaatioissa suurin osa projekteista ei

ole isoja. Tutkimuksen mukaan suurin osa isoista projekteista voitaisiin toteuttaa pienempinä osina normaalilla Scrumilla tai muulla perinteisellä ketterällä menetelmällä, ja kommunikaatio ja synkronointi muiden projektien kesken tulisi hoitaa vain tuoteomistajan tai vastaavan välityksellä. Tutkimuksessa väitetään myös, että SAFe:a on markkinoitu suurille organisaatioille jotka pelkäävät muutosta, illuusiolla että SAFe yhdistää parhaat puolet sekä ketterästä että suurten organisaatioiden byrokraattisesta maailmasta [PG14].

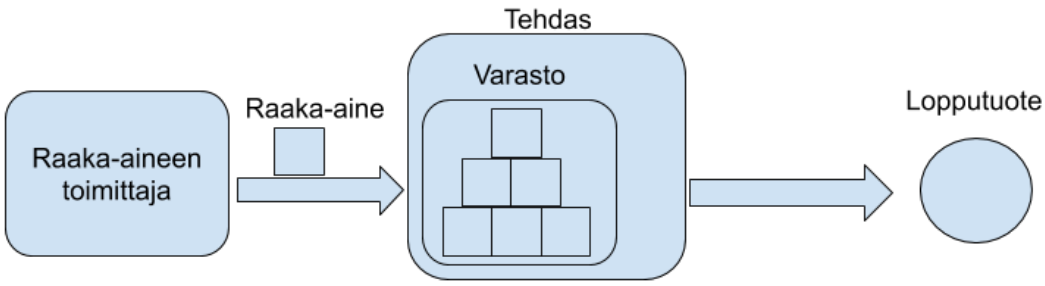
Usein ketterän kehityksen eri menetelmistä puhuttaessa mainitaan myös *Lean-periaatteet*, Lean-menetelmät ja Lean-mallit, kuten *Kanban*. Toyota on ollut erittäin aktiivinen Lean-menetelmien kehittäjä ja saanut menetelmillä hyviä tuloksia aikaan autotehtaissaan [Pop07]. Ohjelmistokehityksessä tunnetuin Toyotan kehittämä Lean-malli on Kanban. Kanbanista kerrotaan tarkemmin alaluvussa 2.1.2.



Kuva 4: Imu-ohjaus. Tehdas tilaa raaka-aineen toimittajalta raaka-ainetta, kun edellinen erä on loppumassa. Tehtaan raaka-aineen varastointitarve vähenee.

Alun perin Lean menetelmät on kehitetty tehdastyöhön tehostamaan työn tehokkuutta, mutta sittemmin näitä samoja menetelmiä on pyritty soveltamaan myös ohjelmistokehityksessä. Lean-periaatteissa pyritään minimoimaan *hukkaa*, eli kaikkea mikä ei tuota arvoa asiakkaalle [Raz16]. Lean perustuu *imu-ohjaukselle*, jossa tarvittavat työvaiheet tehdään vasta kun niitä tarvitaan. Imu-ohjaus on esitettynä kuvassa 4. Tämä eroaa perinteisestä *työntö-ohjauksesta*, jossa jokin ominaisuus tehdään valmiiksi odottamaan seuraavaa

työvaihetta. Työnto-ohjaus on esitettynä kuvassa 5. Lean-mallilla pyritään estämään puolivalmiiden tuotteiden tai ominaisuuksien valmistuminen varastoon [PC12].



Kuva 5: Työntö-ohjaus. Raaka-aineen toimittaja toimittaa tehtaan varastoon raaka-aineita. Tehdas valmistaa näistä raaka-aineista lopputotetta. Työntö-ohjauksessa tehdas joutuu varastoimaan turhaan raaka-ainetta.

2.1 Perinteiset ketterät ja Lean menetelmät

Ketteriä ohjelmistokehityksen malleja on kehitetty useita, mutta tunnetuimpia ja käytetyimpiä näistä ovat XP ja Scrum. Tässä tutkielmassa XP jätetään kokonaan käsittelemättä. Scrumiin perehdytään lisää alaluvussa 2.1.1. Lean-malleista ohjelmistokehityksessä yleisimmin tunnettu on Kanban, johon perehdytään tarkemmin alaluvussa 2.1.2.

Lean-menetelmien soveltamista ohjelmistokehitykseen ovat kehittäneet etenkin Mary ja Tom Poppendieck. He ovat määritelleen *Lean Software Development (LSD)* -mallin, joka pohjautuu monilta osin samoihin teorioihin, joita on käytetty myös ketterän kehityksen pohjana [Pop07]. Ohjelmistokehityksen viitekehyksessä Lean-menetelmillä pyritään vähentämään hukkaa ja byrokratiaa tuotekehityksessä, sekä parantamaan tuotetta nopeasti käyttäjäpalautteen perusteella ennemmin kuin vaatimusmäärittelydokumentaation avulla [PC12].

LSD-malli keskittyy seitsemään pääperiaatteeseen, joita noudattamalla pyritään kehittämään ohjelmistoa nopeammin, paremmin ja pienemmällä budjetilla. LSD:n seitsemän pääperiaatetta ovat:

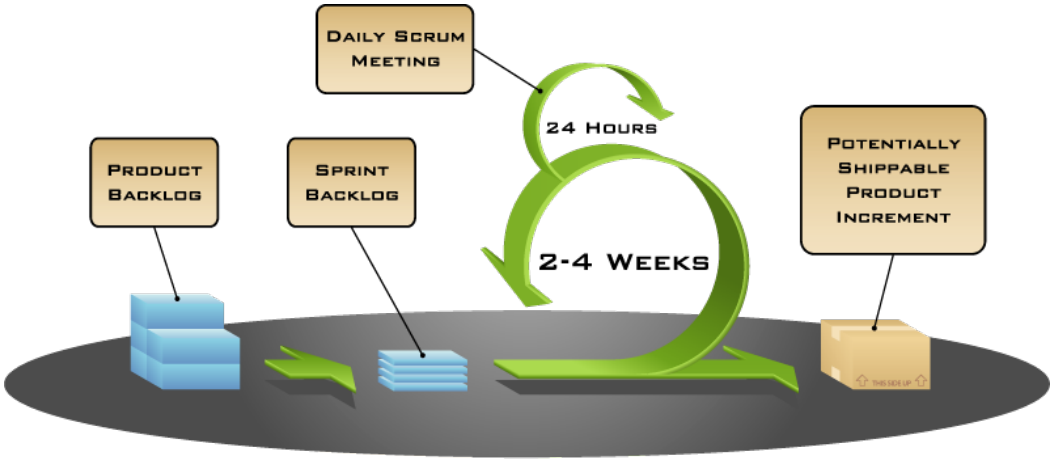
1. Eliminoi hukka
2. Rakenna laadukkaasti
3. Luo tietämystä
4. Lykkää sitoutumista
5. Toimita nopeasti
6. Kunnioita yksilöitä
7. Optimoi kokonaisuus

[Pop07]
Vaikka Lean- ja ketterä kehitys eivät ole täysin sama asia, näitä termejä käytetään kuitenkin toisinaan vastaamaan toisiaan [Wan11]. Erään näkemyksen mukaan Lean on filosofia tai joukko periaatteita, kun taas ketterä kehitys on enemmänkin käytännön tason asioita [Wan11].

2.1.1 Scrum

Scrum on iteratiivinen ja inkrementaalinen ketterän kehityksen malli joka koostuu 1-3 viikon mittaisista sykleistä eli *Sprinteistä*. Scrumiin kuuluu kolme eri roolia *Scrum Master*, *Tuoteomistaja (Product Owner)* ja *kehittäjätiimi*. Scrum Masterin tärkein tehtävä on eliminoida Scrum-tiimin kokemat esteet. Scrum on nykyään käytetyin kehitysmenetelmä ohjelmistokehityksessä [SBS17].

Kuvassa 6 on esitettynä yksinkertainen malli Scrumista. Kaikki toteutettavan tuotteen toteuttamatta olevat ominaisuudet ovat listattuna *Product Backlogille*. Kun asiakas haluaa tuotteeseensa jonkin lisäyksen, tämä lisätään Product Backlogille. Asiakas ja tuoteomistaja vastaavat Product Backlogin priorisoinnista (*Backlog Grooming*).



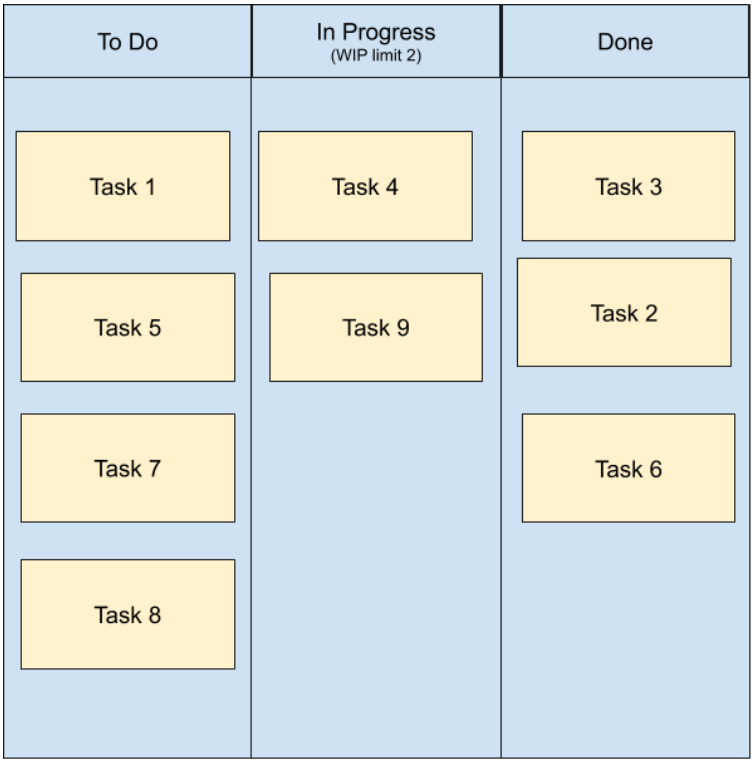
Kuva 6: Scrum [Pay]

Ennen uutta *Sprinttiä* eli kehitysiteraatiota järjestetään *Sprint Planning*-seremonia, johon osallistuu asiakas, kehittäjätiimi, Scrum Master ja tuoteomistaja. Sprint Planningissa muodostetaan seuraavan Sprintin *Sprint Backlog*, jonne kootaan alkavassa Sprintissä toteutettavaksi suunnitellut työtehtävät. Sprintin aikana toteutettavaksi suunniteltuihin työtehtäviin ei voi sprintin aikana tehdä muutoksia.

Sprintin aikana Scrum-tiimi ja Scrum Master kokoontuvat päivittäin *Daily*-seremoniaan, jossa käydään läpi mitä edellisenä päivänä kukin on saanut tehtyä, sekä mitä tänä päivänä kukin tekee. Sprintin päätteeksi tiimi on saanut valmiiksi tuoteinkrementin, joka hyväksytetään asiakkaalla, ja voidaan lisätä olemassa olevaan tuotteeseen. Tämän lisäksi sprintin päätteeksi järjestetään *Sprint Review* eli sprinttikatselmus sekä *retrospektiivi* (*Sprint Retrospective*). Sprinttikatselmuksessa asiakas ja Scrum-tiimi käyvät läpi sprintin aikana valmiiksi saadut tehtävät sekä tehtävät, joita ei ole saatu valmiiksi. Retrospektiivissä käydään läpi tiimin työhön liittyviä kehityskohteita sekä seuraavaan sprinttiin mukaan otettavia tehtäviä [KS].

2.1.2 Kanban

Kanban on alun perin Toyotan 1950-luvulla kehittämä tuotannon ajoitusjärjestelmä. Sana *kanban* on japania ja tarkoittaa mainoskylttiä [AMO13]. Alun perin Kanban-järjestelmissä oli esimerkiksi käyntikorttipaketissa painon mainos vähän ennen käyntikorttipinon pohjaa, jolloin uudet käyntikortit osattiin tilata ennen edellisten loppumista, mutta niitä ei tarvinnut varastoida turhaan.



Kuva 7: Esimerkki Kanban-taulusta. Esimerkin taulussa In Progress -kenttä on rajoitettu kenttä, jossa maksimaalinen työtehtävien määrä (WIP limit) on 2.

Ohjelmistokehitykseen Kanban tuli vuonna 2004, kun Microsoftilla työskennellyt David Anderson avusti pientä IT-tiimiä, joka toimi tehottomasti [AMO13]. Ohjelmistokehityksessä Kanbanin tärkein elementti on Kanban-taulu. Kuvassa 7 on esitettynä Kanban-taulu, joka on jaettu kolmeen sarakkeeseen. *To Do* -sarakkeessa on Kanbanin Backlog, *In Progress* -sarakkeessa meneillään olevat työtehtävät ja *Done*-sarakkeessa valmiit työtehtävät. Kanban taulun sarakkeiden määrää ei kuitenkaan ole rajoitettu, vaan kanban taulussa voi olla niin monta saraketta kuin Kanbania käyttävä tiimi tarvitsee.

Kanbanin perusideat ovat rajoittaa työtehtävien määrää työpanosta vaativissa sarakkeissa (WIP limit), mitata työtehtävien läpikulkuaikaa Kanban-taululla ja visualisoida työtehtävien työnkulkua[AMO13]. Kuvassa 7 tällainen rajoitettu sarake on *In Progress* -sarake. Esimerkkitaulun In Progress -sarakkeessa voi samaan aikaan olla vain kaksi työtehtävää. Mikäli sarakkeessa on jo kaksi työtehtävää, täytyy toinen työtehtävä siis saada siirrettyä sarakkeeseen Done

ennen kuin uusia työtehtäviä voidaan siirtää To Do -sarakkeesta In Progress -sarakkeeseen.

2.2 Skaalatut ketterät menetelmät

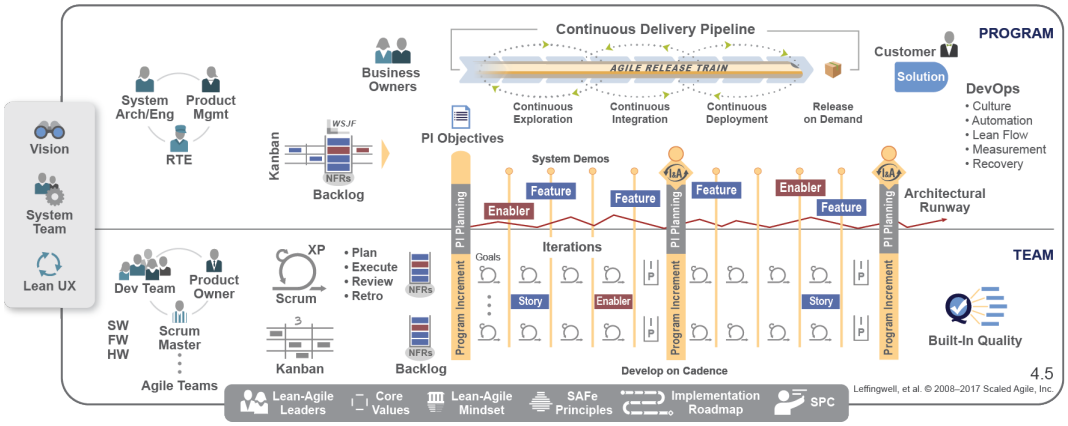
Kun ketterien menetelmien tuomat edut ohjelmistokehityksessä pienten tiimien kanssa havaittiin toimivaksi, samojen menetelmien soveltuvuutta isoille hajautetuille tiimeille alettiin tutkia. Leffingwell kirjoittaa kirjassaan Scaling Software Agility, että useiden ketterän kehityksen pääperiaatteiden on havaittu skaalautuvan hyvin myös isoihin ja hajautettuihin organisaatioihin [Lef07, s. 7]. Haasteena on kuitenkin ollut, että olemassa olevat ketterät ohjelmistokehitysmenetelmät ovat fokusoituneet vain pienen tiimin tarpeisiin [Lef07, s. 88].

Leffingwellin mukaan organisaation kasvaessa, organisaation proseduurit, käytännöt ja mallit kasvavat myös. Tällainen organisaation monimutkaistuminen saattaa hyvin helposti hankaloittaa ketterän kehittämisen filosofian toteuttamista [Lef07, s. 87]. Näihin ongelmiin on kehitetty ratkaisuksi suurille organisaatioille suunniteltuja ketterän kehityksen ohjelmistokehityskehyksiä, kuten SAFe ja *LeSS (Large Scale Scrum)*. Tässä tutkielmassa tutustutaan tarkemmin SAFe-ohjelmistokehityskehykseen ja LeSS jätetään tämän työn ulkopuolelle.

2.2.1 Scaled Agile Framework (SAFe)

Scaled Agile Framework (SAFe) on Dean Leffingwellin ja Scaled Agile Incorporationin alun perin vuonna 2011 esittelemä ketterän ohjelmistokehityksen kehys joka tarjoaa työskentelymenetelmiä organisaation laajuisesti. SAFe tarjoaa roolit, vastuualueet ja toimintamallit ketterään ohjelmistokehitykseen. Kehyksen periaate on tehdä ketterästä ohjelmistokehityksestä sekä suurille projekteille skaalautuvampaa että etenkin suuremmille organisaatioille toimivampaa mallia. Scaled Agile Incorporationin mukaan SAFe skaalautuu yli 50 työntekijän organisaatioista useiden tuhansien henkilöiden organisaatioihin [SAl].

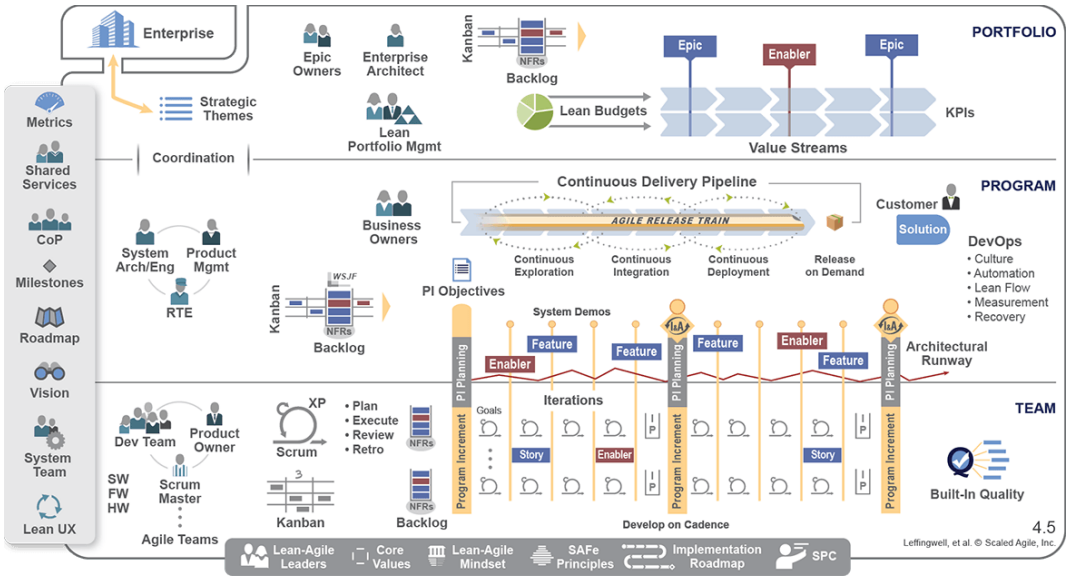
SAFe:sta on olemassa neljä valmista konfiguraatiomallia erikokoisille organisaatioille. *Essential SAFe*, *Portfolio SAFe*, *Large solution SAFe* ja *Full SAFe*. Kaikkien konfiguraatioiden pohjana on Essential SAFe -konfiguraatio eli SAFe:n peruskonfiguraatio. Peruskonfiguraatio on esitetty kuvassa 8, se sisältää *tiimitason (team level)* ja *hanketason (program level)*.



Kuva 8: Essential SAFe -SAFe:n peruskonfiguraatio [SAj].

Kuvassa 9 on esiteltynä SAFe:n Portfolio -konfiguraatio. Tämä taso sisältää SAFe:n peruskonfiguraation lisäksi *portfoliotason (portfolio level)*. Portfoliotaso tarjoaa menetelmiä ja työkaluja muun muassa budjetointiin ja hallinnointiin sekä liiketoiminta-arvojen ymmärtämiseen.

Large Solution SAFe tarjoaa peruskonfiguraation päälle laajojen ratkaisujen tason (large solution level). Tämä taso on tarkoitettu tarjoamaan rooleja ja prosesseja monimutkaisten ja suurien järjestelmiin kehitystyöhön. Laajojen ratkaisujen taso tarjoaa toimintamalleja muun muassa useiden toimitusjunien koordinointiin [SAe].



Kuva 9: Portfolio SAFe -organisaatiokaavio [SAj].

Full SAFe on yhdistelmä Large Solution SAFe:sta ja Portfolio SAFe:sta. Se on tarkoitettu kaikista suurimmille yrityksille, jotka tarvitsevat useita rinnakkaisia SAFe:n konfiguraatioita.

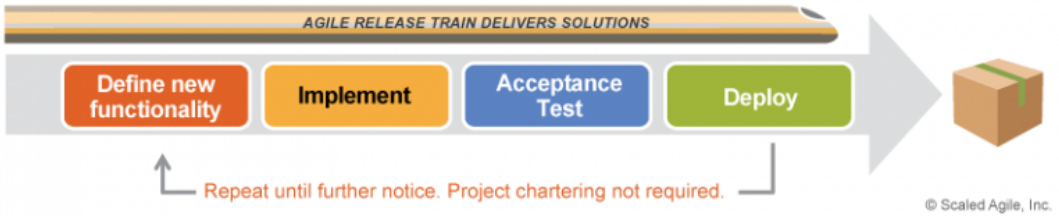
Tässä tutkielmassa haastateltu SAFe-mallia käyttävä tiimi käytti SAFe:n konfiguraatiota, joka pohjautui Portfolio SAFe:n konfiguraatioon. Portfolio SAFe:n konfiguraatio on esitettynä kuvassa 9. Tässä tutkielmassa tutustutaan tarkemmin Portfolio SAFe -konfiguraatioon, muut konfiguraatiot jätetään tutkielman ulkopuolelle.

2.2.1.1 Ketterät tiimit (Agile team) SAFe:ssa

SAFe:n kuten muidenkin ketterien ohjelmistokehityksen pohjalla ovat noin 5 -10 henkilön kokoiset monitaitoiset itseorganisoituvat tiimit. SAFe:ssa ketterä tiimi koostuu *kehitystiimistä*, *Scrum Masterista* ja *tuoteomistajasta* (*Product Owner, PO*). Ketterä tiimi vastaa sille annetun ratkaisun määrittelystä, rakentamisesta, testaamisesta ja mahdollisuuksien mukaan myös tuotantoon viennistä. Tiimin vastuulla on myös itse valita itselleen ketterän kehityksen malli, jota noudattaa. Yleisimmin SAFe:n kanssa käytetään joko Scrum, Kanban tai Extreme Programming (XP) malleja tai näiden mallien osittaista yhdistelmää. [SAb]

2.2.1.2 Toimitusjuna (Agile release train, ART)

Usein SAFe-ohjelmistokehityskehyksen avulla toteutettujen tuotteiden ohjelmistokehityksestä vastaa useampi kuin yksi ketterä tiimi. SAFe-mallissa nämä tiimit työskentelevät synkronoidusti toimitusjunassa. Yhdessä toimitusjunassa työskentelee SAFe:n virallisen dokumentaation mukaan tyypillisesti 50 - 125 henkilöä [SAa]. Toimitusjuna on SAFe:n yksi keskeisimmistä ideoista, ja tämä on yksi suurimmista eroista muihin suurille organisaatioille tarkoitettuihin ketterän kehityksen malleihin verrattuna.



Kuva 10: SAFe toimitusjuna [SAa]

SAFe:n virallisen dokumentaation mukaan toimitusjunan tarkoitus on siirtää kehityksen ajattelumallia pois projektiluontoisesta mallista pitkäkestoisempaan ratkaisuja luovaan malliin [SAb]. Junan tyypillinen sykli on 8-12 viikkoa [SAh]. Tässä ajassa tiimin pitäisi saada *Inkrementti (Product Increment, PI)* valmiiksi. Inkrementti jakautuu useisiin *tehtäviin (tasks)*, joita tiimi tekee itse valitsemansa ketterän kehityksen mallin mukaan, esimerkiksi Scrum-mallia käyttäen iteroiden.

Toimitusjunaa johtaa *Release Train Engineer (RTE)*. RTE:n päätehtävänä on ohjata junan toimintaa tuottamaan mahdollisimman paljon arvoa asiakkaalle. RTE järjestää ennen jokaisen uuden Inkrementin alkua kahden päivän mittaisen *Inkrementin suunnittelupalaverin* (*PI planning*). PI suunnittelupalaverissa toimitusjunan tiimit luovat suunnitelmansa seuraavaa toimitusjunan sykliä sekä tulevaa Inkrementtiä varten. Palaverissa käydään läpi myös koko SAFe-organisaation näkemystä lähitulevaisuuden ohjelmistokehitystarpeista ja toimintasuunnitelmaa seuraavan iteraation osalta [SAg]. RTE toimii myös kommunikoinnin välikätenä ohjelmistokehitystiimin ja hallinnon välillä [SAi].

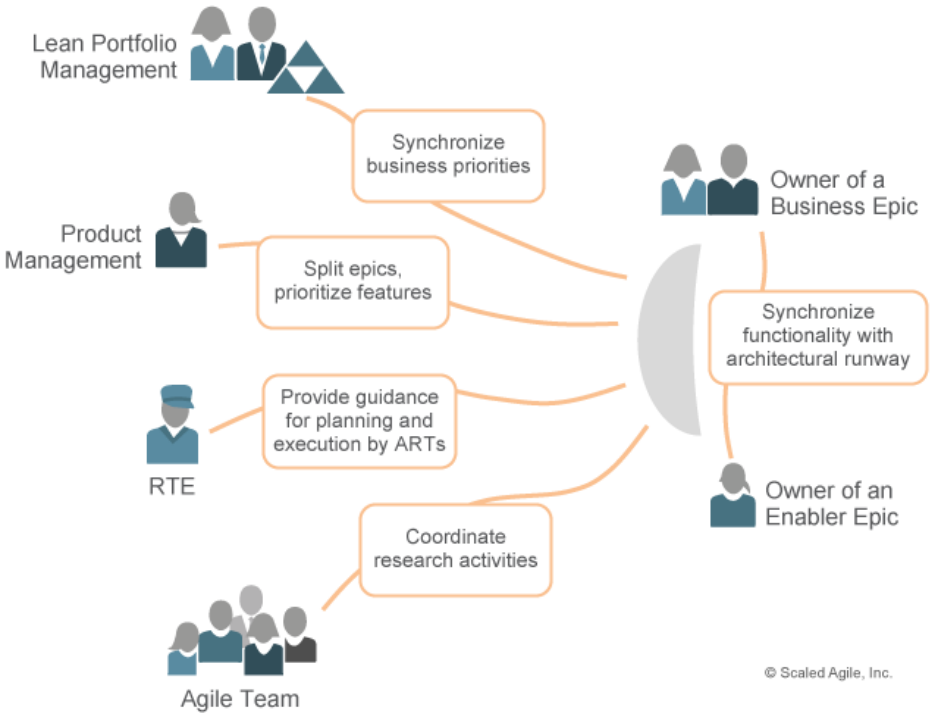
2.2.1.3 Arkkitehtuuri

Arkkitehtuuri on SAFe:ssa jaettu kahdelle eri tasolle. Program-tasolla toimii useille tiimeille jaettuja systeemi- ja rarkaisuarkkitehteja. Nämä arkkitehdit vastaavat tarkemmin ratkaisujen arkkitehtuurista sekä rajapintojen toimivuudesta muiden projektin tiimien kesken [SAk].

Portfolio-tasolla toimivat Enterprice -arkkitehdit, jotka vastaavat sovelluksen suurista linjoista. Enterprice arkkitehdit työskentelevät yhdessä systeemi- ja ratkaisuarkkitehtien kanssa [SAc].

2.2.1.4 Epic Owner

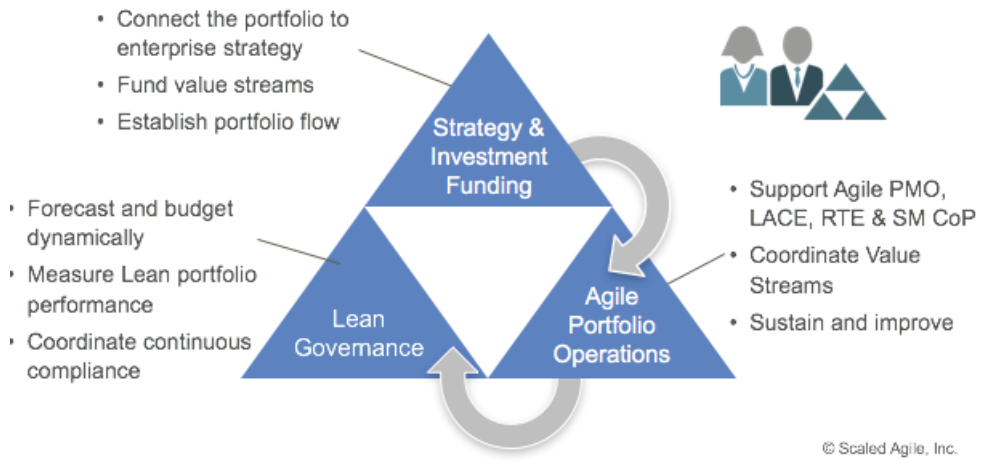
SAFe:ssa jokaiselle *Kehitysaihiolle (Epic)* on määrätty *Epic Owner* vastaamaan kyseisen kehitysaihion läpikulusta. Epic Owner määrittelee kehitysaihiolle MVP:n eli pienimmän toimivan tuotteen määritelmän [SAd]. Epic Owner kommunikoi kaikkien kehitysaihion parissa työskentelevien kanssa. Kuvassa 11 on esitettynä Epic Ownerin kommunikointikanavia eri roolien kanssa.



Kuva 11: Epic Owner -roolin kommunikaatiosuhteet muihin rooleihin [SAd].

2.2.1.5 Lean salkunhallinta

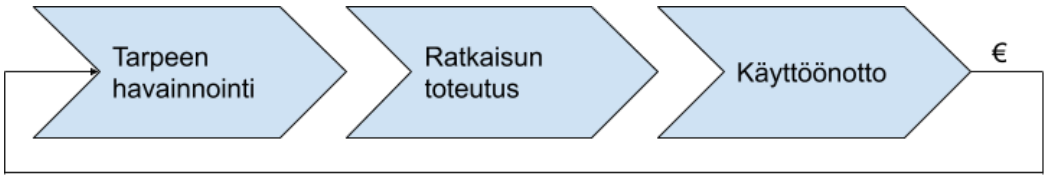
SAFe:n portfoliotason tehtävistä ehkä tärkein on Lean salkunhallinta. Lean salkunhallinnan tavoitteena on luoda korkeimman tason päätöksiä ja talousvastuita SAFe-portfolion tuotteille. Salkunhallinnan parissa työskentelee henkilöitä jotka ovat vastuussa yrityksen taloushallinnosta, johdosta, strategisesta suunnittelusta sekä operoinnista. Lean salkunhallinta koostuu kolmesta päävastuualueesta: *Strategia ja investointien rahoitus*, *Ketterän portfolion toimet* ja *Lean hallinto* [SAf]. Lean salkunhallinnan pääkohdat on esitettynä kuvassa 12.



Kuva 12: SAFe:n Lean salkunhallinnan päävastuualueet [SAf]

Lean salkunhallinnan vastuualueista tärkein on strategia ja investointien rahoitus. Tässä vastuualueessa Epic Owner yhdessä Enterprice arkkitehtien, tuoteomistajien sekä yhtiön johdon kanssa suunnittelee tuotteen portfoliota yhtiön strategian mukaan. Portfolion tuotantojärjestystä käydään läpi, priorisoidaan kehitysaihioiden (Epic) järjestys Portfolio Kanbanin backlogille sekä sovitaan rahoituksesta *arvovirroille*.

Arvovirrat ovat syklisiä portaittain eteneviä tehtäviä, jotka luovat asiakkaalle arvoa [SAf]. Kuvassa 13 on esitettynä kuvitteellinen arvovirta tuotteen ominaisuudelle. Näiden tehtävien tarkoituksena on varmistaa, että tuotannossa on vain se määrä tuoteaihioita, joka pystytään yrityksen kehityskapasiteetilla toteuttamaan.



Kuva 13: Arvovirta. Kuvassa € kuvaa asiakkaan kokemaa arvoa, jonka tuottaminen on arvovirran keskeinen tavoite.

Ketterissä portfolion toimissa on tarkoituksena koordinoida arvovirtoja toimimaan mahdollisimman tehokkaasti. Tämä toteutetaan pitämällä yllä jatkuvaa kommunikointia tuotannon ohjausryhmän, Scrum Mastereiden sekä RTE:n kesken. Kommunikoinnilla pyritään ylläpitämään jatkuvaa ketterien menetelmien parantamista, jonka avulla tuotantoa saadaan tehostettua [SAf].

Lean salkunhallinnan tarkoituksena on ennustaa pitkän aikavälin tuotannon budjettitarvetta, mitata tuotannon tehokkuutta sekä koordinoida ja valvoa yrityksen sekä alaa mahdollisesti koskevien säännösten noudattamista [SAf].

2.3 Tunnetut haasteet ohjelmistokehityksessä

John Erickson et al. kirjoittavat, että ketterän kehityksen ideologian pohja on siinä, että perinteisestä ohjelmistokehityksestä poistetaan mahdollisimman paljon raskautta, jotta voidaan reagoida nopeasti ympäristön muuttuviin tilanteisiin, kuten muuttuviin käyttäjävaatimuksiin tai projektin määräaikojen muutoksiin. Erickson et al. mukaan perinteinen ohjelmistokehityksen malli on liian hidas vastaamaan tällaisiin muutoksiin tarpeeksi nopeasti, vaikka usein niin markkinoidaankin [ELS05].

2.3.1 Skaalaamisen haasteet

Vaikka SAFe on ketterän kehittämisen ohjelmistokehityskehys, on useissa tutkimuksissa havaittu ketterien mallien soveltuvuudessa suurille organisaatioille olevan haasteita. SAFe-mallin tuominen isoon organisaatioon on joidenkin mielestä vaikuttanut pikemminkin siirtymiseltä pois ketterästä kehityksestä, kuin ketterän kehityksen parantamiselta. Tämän väitteen taustalla ovat useat erityisen hyvin ketterään kehittämiseen perehtyneet henkilöt, kuten Ken Schwaber (toinen Scrum-mallin kehittäjistä), Ron Jeffries (yksi XP-mallin kehittäjistä) ja Stephen Denning (Scrum Alliancen hallituksen jäsen) [PPL18]. Erityisesti SAFe:a on kritisoitu sen tavasta pakottaa organisaation malli ketterän kehityksen ideologian horisontaalisesta organisaatiomallista takaisin vertikaaliseen ja hierarkkiseen byrokratiamalliin, jolloin tuottavuus mahdollisesti kärsii [Ste15].

On väitetty myös, että SAFe tappaisi koko ketterän kehityksen hengen [PG14].

Tämä sama haaste on havaittu RAY:lla ohjelmistokehityksestä tehdyssä tutkimuksessa. Tutkimuksessa havaittiin RAY:n ottaessa SAFe-ohjelmistokehityskehystä käyttöön vuonna 2013, että mikäli budjetointi-, konseptointi- ja päätöksenteko-osastot eivät omaksu ketterän kehityksen käytäntöjä, niin ketterää ohjelmistokehitystä ei saada toimimaan tulevaisuuden vaatimusten mukaisesti [Col15, s. 113]. Myös muissa ketterää ohjelmistokehittämistä koskevissa tutkimuksissa on havaittu, että suuriin ja monimutkaisiin projekteihin on hankala ottaa käyttöön ketterän kehityksen metodeja [DD08]. On myös havaittu, että yksi tärkeimmistä, mutta samalla vaikeimmin toteutettavista muutoksista ketterään kehitykseen siirryttäessä on organisaation toimintakulttuurin muutos [Sub10]. On esitetty myös hypoteeseja, että ketterät menetelmät eivät välttämättä aina edes olisi parhaita menetelmiä tehdä ohjelmistokehitystä suurissa projekteissa [DD08].

Martin Kalenda on tutkinut pro gradu -tutkielmassaan systemaattisen analyysin avulla tieteellisiä julkaisuja suurten organisaatioiden ketterien menetelmien käyttöön ottamisesta, ja havainnut että eräs yleinen haaste ketterien menetelmien käyttöön ottamisessa on organisaation työntekijöiden muutosvastarinta. Kalendan havaintojen mukaan ketterässä kehittämisessä keskijohdon ei ole enää tarkoitus ohjata suoraan tiimien tekemistä. Tällainen vastuumuutos perinteiseen toimintamalliin verrattuna saattaa hämmentää keskijohdon työntekijöitä ja aiheuttaa muutosvastarintaa, kun työntekijät pelkäävät, että heidän työpanostansa ei tarvita enää. Kalenda on havainnut myös, että monet keskijohdon päälliköt eivät välitä tiimien tekemistä päätöksistä tai visioista ohjelmistokehityksen suhteen, vaan alkavat mikromanageroida tiimejä. Tämä aiheuttaa useita ongelmia, kuten vähentää tiimin jäsenten kiinnostusta osallistua toimintansa kehittämiseen [Kal17]. Misra et al. ovat havainneet muutosvastarintaa ketterien menetelmien käyttöönotossa myös kehittäjien keskuudessa. Heidän mukaansa muutosvastarintaa kehittäjien keskuudessa aiheuttaa etenkin vanhat tavat toteuttaa ohjelmistokehitystä perinteisessä hierarkkisessa organisaatiomallissa [Sub10].

2.3.2 Kommunikoinnin haasteet

Leffingwell kirjoittaa kirjassaan Scaling Software Agility, että yksi ketterän kehityksen tiimin onnistumisen edellytyksistä on toimiva kommunikointi [Lef07]. Leffingwell viittaa myös Simonsin artikkeliin, jossa todetaan, että hajautetun tiimin perusongelmana on kompleksisen projekti- ja prosessispesifisen tiedon jakamisen ongelmat erillään olevien tiimien välillä. Simons kirjoittaa myös, että ongelmaa voidaan vähentää yleisesti sovituilla tiimirakenteilla ja kurinalaisella viestimisellä, mutta nekään eivät korvaa täysin paikan päällä tapahtuvaa vierailua [Lef07][Sim06]. Leffingwellin mielestä hajautettujen tiimien välistä kommunikointia voidaan parantaa mm. usean kokoushuoneen telekonferenssein [Lef07]. Myös *Ketterän ohjelmistokehityksen julistuksen* (*Agile manifesto*) takana olevissa periaatteissa todetaan seuraavasti:

> *Tehokkain ja toimivin tapa tiedon välittämiseksi kehitystiimille ja tiimin jäsenten kesken on kasvokkain käytävä keskustelu.*
>
> *– [BBvB+]*

3 Tutkimuksen toteutus

Tutkimus toteutettiin haastatteluihin perustuvana laadullisena tapaustutkimuksena. Tapaustutkimukset ovat yleinen tutkimuskeino etenkin humanistisilla aloilla, mutta se toimii hyvin myös ohjelmistokehityksen tutkimisessa, sillä ohjelmistokehitys on monitieteellinen ja kehittyvä ala [RH08]. Tapaustutkimus sopii hyvin tutkimusmetodiksi monenlaisiin ohjelmistokehitysalan tutkimuksiin, sillä monia nykyajan ilmiöitä on vaikea tutkia eristetysti. Tapaustutkimukset eivät tuota samanlaisia syy-yhteyksiä, kuin kontrolloidut laboratoriotutkimukset. Sen sijaan tapaustutkimukset tarjoavat syvempää ymmärrystä tutkittavasta ilmiöstä [RH08].

Laadullisen tutkimuksen aineistonkeruun voi jakaa kolmeen eri menetelmään: 1) suoraan menetelmään, jossa tutkija on suorassa yhteydessä tutkimuskohteeseen, kuten teemahaastattelut; 2) epäsuoraan menetelmään, jossa tutkija kerää aineistoa ilman interaktiota tutkittavan kohteen kanssa; 3) riippumattomaan analyysiin valmiiksi kerätyn aineiston perusteella [RH08]. Tämä tutkimus on toteutettu suoralla menetelmällä puolistrukturoiduilla teemahaastatteluilla.

3.1 Aineiston keruu

Tutkimuksessa käytetty aineisto perustuu haastatteluihin, sekä niistä tehtyihin litterointeihin. Tutkimuksen haastattelukeinona käytettiin puolistrukturoitua teemahaastattelua. Teemahaastattelussa oleellisinta on, että haastattelu etenee yksityiskohtaisten kysymysten sijaan tiettyjen keskeisten teemojen varassa. Hausjärven ja Hurmeen mukaan teemahaastattelu tuo tutkittavien äänen kuuluviin ja ottaa huomioon ihmisten tulkinnat asioista. Teemahaastattelusta puuttuu strukturoiduille haastatteluille luonteenomainen tarkka muoto ja järjestys [Hir08]. Haastattelurungon kaikkia kysymyksiä ei siis välttämättä tarvitse kysyä ja haastattelija voi esittää haastattelurungon kysymyksiin tarkentavia lisäkysymyksiä. Haastattelurungon kysymyksiä ei myöskään tarvitse puolistrukturoidussa teemahaastattelussa esittää haastattelurunkoon kirjatussa

järjestyksessä. Puolistrukturoitu teemahaastattelu on yleinen tutkimusmetodi tapaustutkimuksissa [RH08].

Haastatteluiden alussa tutkijan tulee kertoa haastateltavalle tutkimuksen tarkoitus ja kuinka vastauksia tullaan käsittelemään. Tämän jälkeen tutkija voi aloittaa haastattelurungon läpikäynnin. Vaikka haastattelujen aikana tutkija tekisi muistiinpanoja, haastattelut on silti hyvä nauhoittaa [RH08]. Tutkimusta varten tehdyt haastattelut nauhoitettiin, sekä haastatelluille kerrottiin haastattelun alussa mitä aihetta tutkimus koskee, sekä miten haastattelumateriaalia tullaan käsittelemään.

Kaikki tutkimukseen haastatellut henkilöt työskentelevät asiantuntijatehtävissä ja haastatteluihin pyrittiin saamaan henkilöstöä mahdollisimman monesta eri työroolista. Haastateltavista projektitiimeistä toisen kotipaikka on Helsingissä ja toisen Tampereella. Molemmissa tiimeissä työskentelee henkilöitä myös muissa sijainneissa. Kysymysrungot olivat erilaiset SAFe-projektin parissa työskenteleville ja Scrum-projektin parissa työskenteleville. Myös eri työtehtävissä työskenteleville muokattiin haastattelurungon kysymyksiä vastaamaan enemmän kunkin työtehtävän osa-aluetta. Haastattelujen pääteemat olivat kuitenkin kaikille samat.

Kaikilta haastatelluilta kysyttiin seuraaviin aihealueisiin liittyviä kysymyksiä:

- Haastatellun työntekijän tiimin koko, sekä vastaajan mielipide optimaalisesta tiimin koosta.
- Minkälaisessa työympäristössä haastateltu työskentelee ja miten haastateltu kokee työympäristön vaikuttavan työntekoonsa.
- Työskenteleekö tiimi hajautetusti eri toimipisteissä.
- Kuinka paljon haastateltu osallistuu kokouksiin, joissa joku osallistuja on etäyhteydellä mukana, ja kuinka tämä vaikuttaa haastatellun mielestä osallistumiseen.
- Ketkä haastatellun organisaatiossa päättävät sovelluksen kehityksen linjoista.

- Minkälainen tuotantoympäristö sovelluksella on ja miten tuotantoon vienti on järjestetty. Näiden vastausten perusteella kyseltiin haastatellun mielipidettä toisenlaisesta tuotantoympäristöstä ja tuotantoon viennin prosessista.
- Mitä ohjelmistokehitysmenetelmää ohjelmoijatiimi haastatellun mielestään käyttää, onko se toimiva, mitkä sen haitat ja hyödyt ovat ja voisiko sitä jotenkin parantaa.
- Kuinka tarkasti tiimi haastatellun mielestä noudattaa valittua ohjelmistokehitysmenetelmää.

Tämän lisäksi tuoteomistajilta ja arkkitehdeilta kysyttiin vielä tarkentavia kysymyksiä projektin suunnitteluvastuista ja arkkitehtuurin suunnittelusta.

Haastattelut suoritettiin yrityksen neuvottelutiloissa kunkin haastateltavan normaalissa toimipaikassa. Kaikilta haastatelluilta ei kysytty kaikkia kysymyksiä, sillä osa kysymyksistä oli tietyille haastateltaville irrelevantteja ja osa haastateltavista vastasi tuleviin kysymyksiin jo ennen niiden esittämistä. Haastatteluissa esitettiin tarvittaessa tarkentavia jatkokysymyksiä. Haastattelut nauhoitettiin ja litteroitiin mahdollisimman nopeasti haastattelun jälkeen. Haastattelut suoritettiin joulukuun 2018 ja toukokuun 2019 välisenä aikana. Aluksi haastateltiin SAFe-mallia käyttävä tiimi ja tämän jälkeen Scrum-mallia käyttävä tiimi. Ennen haastattelua haastateltaville kerrottiin tutkimuksen aiheesta ja kuinka vastauksia tullaan käsittelemään.

Aineiston tulkintastrategiaksi valittiin laadullinen eli kvalitatiivinen tutkimus. Laadullinen tutkimus on yleinen tulkintastrategia joustavissa tapaustutkimuksissa [RH08]. Hirsjärven mukaan kvalitatiivinen tutkimus tuo esille tutkittavien havainnot tilanteista ja antaa mahdollisuuden heidän menneisyyteensä ja kehitykseensä liittyvien tekijöiden huomioimisen [Hir08]. Näin ollen kvalitatiivinen tulkinta toimii hyvin asiantuntijatehtävissä toimivien henkilöiden kanssa, jolloin heidän omat näkemyksensä haastateltavasta aiheesta saadaan hyvin esille.

Kvalitatiiviseen tutkimukseen kuuluu kaksivaiheinen aineiston analysointi: hypoteesien generointi ja hypoteesien varmistaminen. Hypoteesien generoinnissa tutkimusaineiston perusteella luodaan hypoteeseja, joiden todenperäisyys varmennetaan hypoteesien varmistamisvaiheessa. Kvalitatiivisessa tutkimuksessa on oleellista etsiä ja organisoida laajasta aineistomäärästä oleelliset asiat, sekä muodostaa näistä selkeät päättelyketjut. Luottamuksen tukemiseksi kerätystä aineistosta täytyy jakaa relevantteja otoksia, kuten lainauksia haastatteluista [RH08].

Haastatteluja suoritettiin yhdeksän kappaletta. Haastateltavien pienestä määrästä johtuen kvantitatiivista tutkimusta ei olisi tällä otannalla pystytty tekemään. Osa haastateltavista vastasi kysymyksiin hyvin laajasti, kun taas osa haastateltavista vastaili vain muutamilla sanoilla. Näin ollen haastattelujen kestoajoissa oli suuria eroja.

3.2 Tutkimuksessa mukana olleet tiimit

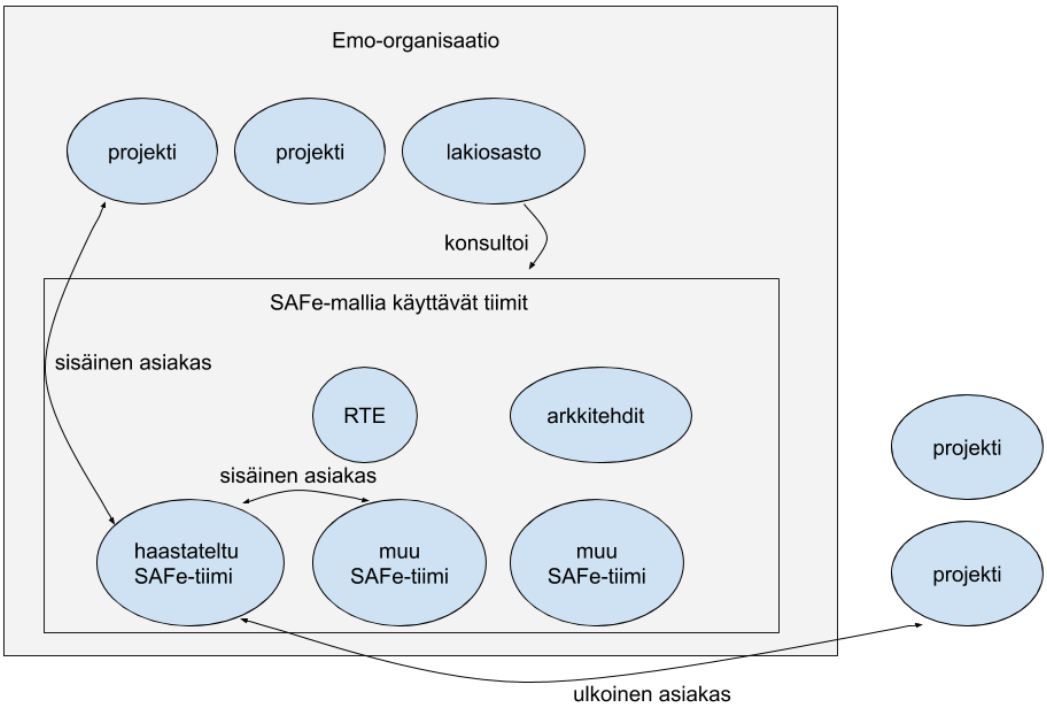
Valitsin tutkimukseen mukaan kaksi eri toimintatavoin toimivaa tiimiä, jotka toteuttivat keskenään kilpailevaa, vaatimuksiltaan identtistä palvelua. Tällä valinnalla pyrittiin poistamaan toteutettavan tuotteen erilaisista vaatimusmäärittelyistä mahdollisesti johtuvat erot tuotteen kehitysprosessissa. Vaikka haastattelujen aikaan työskentelin yhtiön palveluksessa, en ollut muiden töideni osalta yhteistyössä kummankaan tiimin kanssa.

Haastattelujaksojen aikana organisaatiossa tehtiin organisaatiouudistusta, minkä johdosta tiimien on määrä yhdistyä. Yhdistymisen valmistelut aloitettiin jo Scrum-tiimin haastatteluiden aikana, mutta yhdistymistä ei ehditty toteuttaa haastatteluiden aikana. Scrum-tiimin haastattelujen aikana kävi ilmi, että SAFe-tiimin tekemä tuote lopetetaan ja Scrum-tiimin tekemän tuotteen kehitystä jatketaan. Haastattelujen keräämisen aikaan tuotetta ei ollut vielä otettu tuotantokäyttöön. Samaan aikaan organisaatiouudistuksessa uudistettiin myös työtapoja ja SAFe-ohjelmistokehityskehyksen käytöstä päätettiin luopua.

SAFe-kehyksen tilalle päätettiin ottaa Scrum-johdannainen organisaation itse kehittelemä ketterän kehityksen malli.

Kuvassa 14 on esitettynä SAFe-tiimin organisaatiomalli ja kuvassa 15 on esitettynä Scrum-tiimin organisaatiomalli. Tutkimuksessa mukana olleet tiimit eivät alun perin tehneet yhteistyötä, eikä tiimeillä ollut suoria yhteyksiä toisiinsa. Molemmat tiimit konsultoivat emo-organisaation lakiostastoa juridisissa kysymyksissä. Kumpikin tiimi oli varautunut palvelemaan tuotteellaan sekä sisäisiä, että ulkoisia asiakkaita. Vaikka Scrum-tiimi toimi omassa osakeyhtiössään, sisäiseksi asiakkaaksi voidaan laskea myös emo-organisaatiossa toimivat projektitiimit, sillä Scrum-tiimi käytti myös emo-organisaation sisäisiä viestintäkanavia sekä toimi osittain emo-organisaation tiloissa.

3.2.1 SAFe-tiimi



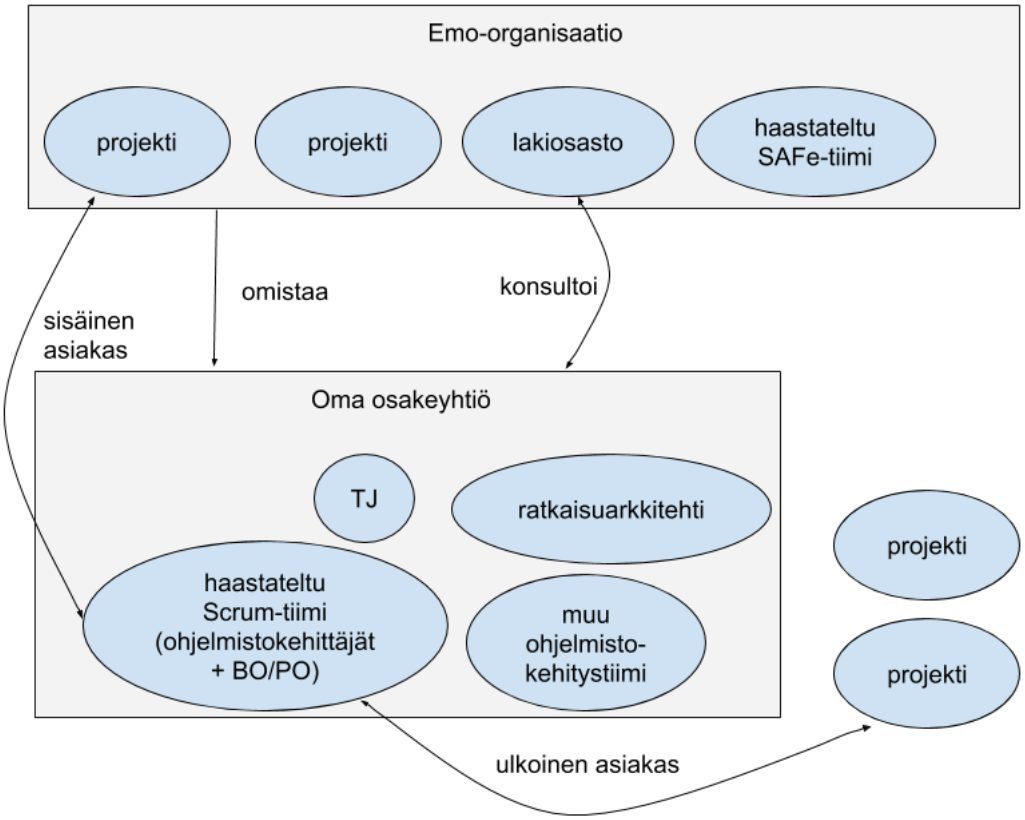
Kuva 14: SAFe-tiimin yksinkertaistettu organisaatiokaavio

SAFe-tiimi työskenteli SAFe-mallin mukaisessa organisaatiossa, jossa samassa toimitusjunassa työskenteli viisi rinnakkaista tiimiä. Tutkimukseen osallistuneessa tiimissä työskenteli täysipäiväisesti kolme kehittäjää, testaaja ja Product Owner. Haastatteluissa kehittäjät nimesivät tiiminsä jäseniksi myös osa-aikaisesti tiimin kanssa työskentelevät Scrum Masterin sekä arkkitehdin. Tämän lisäksi organisaatiossa oli kaikille SAFe:n portfolio-konfiguraatiossa määrittelemille työtehtäville määritelty vastuuhenkilö. SAFe-tiimistä haastateltiin RTE:tä, sovellusarkkitehtia, raitkaisuarkkitehtia sekä kahta ohjelmistokehittäjää. SAFe-tiimin tuottama ohjelmisto suoritettiin keskitetyn ylläpidon ylläpitämässä suurtietokoneessa. Ohjelmiston tuotantoon viennin prosessista vastasi oma tiiminsä, jolle tutkimuksessa mukana ollut tiimi toimitti valmiin version ohjelmistostaan.

SAFe-tiimi käytti tuotekehitystiimin kanssa ohjelmistonkehitysmenetelmänä Scrumia. He määrittelivät toteuttavansa Scrumia "hyvin tarkasti"[ohjelmistokehittäjä 2]. Haastatellut kertoivat organisaation noudattavan myös SAFe-mallia tarkasti.

> *Pidetään kaikki määrätyt sprint planit, dailyt, retrot, reviewit ja tällaset käytänteet*
>
> *– Ohjelmistokehittäjä 1*

3.2.2 Scrum-tiimi



Kuva 15: Scrum-tiimin yksinkertaistettu organisaatiokaavio

Toisessa tutkituista tiimeistä työskenteli täysipäiväisesti kaksi kehittäjää sekä tuoteomistaja. Kaikilla tiimin jäsenillä oli pitkä kokemus ketterien menetelmien käytöstä työssään. Scrum-tiimistä haastateltiin tuoteomistajaa, ratkausarkkitehtia sekä kahta ohjelmistokehittäjää. Tiimi oli saanut valita ohjelmistokehitysmenetelmänsä ja määritteli käyttämänsä ketterän kehityksen mallin Scrumiksi. Haastatellun tiimin jäsenet kertoivat käyttävänsä tätä mallia hyvin väljästi, sekä mallin muuttuneen projektin aikana entistä väljemmäksi. Haastatteluiden perusteella käytetty malli on lähempänä Kanban-tyylistä mallia, kuin Scrum-tyylistä mallia. Kysyttäessä tuoteomistajalta, että onko malli hänen mielestään lähempänä Scrum-tyylistä vai Kanban-tyylistä kehittämistä, hän päätyi siihen,

että kehittäminen oli enemmän Kanban-tyylistä. Ratkaisuarkkitehdin mukaan kehityksessä käytetty malli oli Scrumin ja Kanbanin yhdistelmä.

Projektin alussa kehitystä tehtiin kahden viikon sprinttisyklillä, mutta sittemmin tästä on väliaikaisesti luovuttu. Tiimin näkemyksen mukaan kahden viikon sprinttisykli rajoitti jatkuvasti muuttuviin vaatimusmäärittelyihin vastaamisen joustavuutta. Tiimillä ei ole alun perin ollut käytössä mitään Scrumin normaaleja seremonioita, kuten säännöllisesti järjestettäviä *Dayleja*, *Demoja* tai *Retrospektiivejä*. Tiimi on kertonut sprinttisykliä käyttäessään noudattavansa kahden viikon välein toistuvia *Backlog grooming*-palavereja, jotka ovat olleet eräänlaisia Demo-palavereiden ja seuraavan sprintin suunnittelupalavereiden yhdistelmiä (*Sprint planning)*. Retrospektiivejä on järjestetty epäsäännöllisin väliajoin. Näiden lisäksi on pidetty viikoittain toistuvia *weekly*-palavereja, joissa on käyty läpi menneen ja tulevan viikon asioita.

Scrum-tiimin tuottama ohjelmisto suoritettiin Amazon AWS ympäristössä palvelussa, joka suorittaa virtuaaliympäristössä *kontteihin* (*container*) pakattuja ohjelmistoja. Scrum-tiimi vastasi pääsääntöisesti itse tuotantoympäristönsä ylläpidosta sekä tuotantoon viennistä.

4 Tutkimuksen tulokset

Tässä luvussa käytetyt termit ovat haastateltujen itse määrittelemiä termejä. Luvussa 5 tulkitaan haastateltujen näkemyksiä käyttämistään termeistä. Luku on jaettu aliluvuiksi siten, että aluksi käydään läpi SAFe-tiimin kokemuksia ketterästä kehittämisestä suuressa organisaatiossa, tämän jälkeen käydään läpi haastateltujen tiimien kokemuksia hajautetun tiimin kanssa toiminnasta. Kolmas aliluku käsittelee yhtäläisyyksiä ja eroja SAFe- ja Scrum-tiimin toimintatavoissa, Neljäs aliluku käsittelee tiimien tulkintaeroja tietoturvavaatimuksista. Viidennessä aliluvussa käydään läpi tiimien valitsemien ohjelmistokehitysmallien käytännön eroja, jonka jälkeen kuudennessa aliluvussa käydään läpi tiimien kokemuksia valittujen ohjelmistokehitysmallien hyödyistä ja haitoista. Seitsemännessä aliluvussa käsitellään tiimien käsityseroja ketterästä kehittämisestä.

4.1 SAFe-tiimin kokemat haasteet skaalaamisessa

Vaikka SAFe on ketterän kehittämisen kehys, on SAFe-tiimi havainnut organisaatiossa haasteita ketterän kehittämisen filosofian toteuttamisen kanssa. Molemmat haastatellut SAFe-tiimin kehittäjät kertoivat organisaation kehittävän ohjelmistoja ketterästi heidän näkökulmastaan. Kuitenkin muut SAFe-tiimistä haastatellut kertoivat ketterän kehittämisen haasteista. Kehittäjien näkökulmaa ketterästä kehittämisestä kuitenkin tukee SAFe-tiimin ratkaisuarkkitehdin näkemys, jonka mukaan kehittäjätiimi toimii ketterästi, ja vasta ylemmällä tasolla ketteryys kärsii.

Portfoliotasolla se ei kyllä kauhean ketterää se touhu ollut. En tiedä voiko se koskaan ollakaan siellä niin kauhean ketterää. Yksi Epic kesti tavallaan puoli vuotta ja se syö yhden-kahden tiimin kaistan sen aikana. Epicien käsittelymallit näyttää kyllä enemmän vesiputouksilta. Kun tulee joku uusi idea, niin ensin yritetään hajottaa sitä osiin että saadaan edes Featuret tunnistettua siitä ja siellä on ne tietyt stepit. Tiimillehän toi menee oikeastaan vasta siinä vaiheessa kun Featurella on DoR:ri. Eli se on tipahtanut vesiputouksessa tarpeeksi pitkälle. Sen jälkeen päästään siihen Scrum-maailmaan aidosti käsiksi.

– SAFe-projektin ratkaisuarkkitehti

Osa SAFe-tiimin kokemista ongelmista johtui myös ketterän filosofian omaksumisen puutteesta organisaation laajuisesti. Yhtenä mainittuna ongelmana oli osan organisaation päätöksiä tekevien osastojen sitoutumattomuus SAFe-mallin noudattamiseen. SAFe-mallin portfoliotasolle annettavien strategisten teemojen sijaan portfolio-tasolle tuotiin suoraan vaatimuksia ominaisuuksista ja toiminnoista portfolion backlogin ohi.

Se et mikä sen portfolion rooli oli, niin oli mun mielestä vähän niin ja näin. Se ei niinkun toimi puhtaasti SAFe-moodissa silloin, jos ei itse saa päättää siitä, mikä on se tärkeysjärjestys missä hommat pitää tehdä. Enemmän se ongelma varmaan tuli niistä organisaation vanhoista rakenteista, kun siellä oli monta tasoa mistä päätöksentekoa tehtiin sen portfolio-tason yläpuolella. Se ei oo sillon kauhean selkeetä. Sitten taas, kun niitä tarpeita tulee enemmän alhaalta ylöspäin, että kun joku huomaa että tarvitaan tommonen tunnistamishomma, niin se ei taas vastaa sitä mitä siellä ylhäällä on taas päätetty.

– SAFe-projektin ratkaisuarkkitehti

SAFe-tiimin organisaatiossa on ollut myös haasteita monimutkaisten ja suurten järjestelmien ylläpidon ja tuotantoon viennin kanssa, sillä uusien ominaisuuksien tuotantoon viennissä meni useita viikkoja aikaa. SAFe-tiimin tuottaman ohjelma oli tarkoitus toimia organisaation keskitetyssä ympäristössä, jonka ylläpidosta vastaa dedikoitu tiimi.

> *Sen lanseerauksen - releasoinnin - ja tuotannon välissä on paljon asioita. Se on monimutkaista. Siinä pitää olla ihan oma porukkansa.*
> *– SAFe-projektin RTE*

4.2 Kommunikoinnin haasteet

Haastateltu sovellusarkkitehti kertoo SAFe-projektissa erään tiimin työskentelevän Virossa, ja heillä olevan haasteita tiimin kanssa kommunikoinnin kanssa.

> *Meillä on tälläkin hetkellä sellainen tilanne päällä, että me ei suoraan nähdä mitä se tiimi on siellä tekemässä, onko sillä ongelmia. Me vaan katsotaan, että JIRA:ssa tiketit etenevät todella hitaasti ja koodaaja ei oikein kommentoi mitään asioihin. Niin me nyt mietitään että tuodaan se kaveri tänne. Istutetaan ne tänne, niin me nähdään missä ne menee ja jos niillä on ongelmia, niin me ollaan heti siinä ja pystytään jeesaamaan niissä asioissa.*
> *– Sovellusarkkitehti SAFe-projektissa*

Myös Scrum-tiimillä oli ennakko-oletuksia hajautetun tiimin työskentelyn suhteen. He kuitenkin olivat ratkaisseet ongelman pitämällä jatkuvaa kommunikointia yllä nykyaikaisilla viestintävälineillä. Kysyttäessä hajautetun tiimin vaikutuksesta tiimin väliseen kommunikointiin ja työtehokkuuteen, Scrum-tiimiltä saatiin seuraavanlaisia vastauksia.

En koe, että siitä on suoranaista haittaa. Se vaatii kypsemmän tiimin ja tietynlaista tahtotilaa. Tiimin täytyy itsekin haluta sitä ja tiimin täytyy ylikommunikoida. Uskon osittain semmoiseen James Goslinin viisauteen historiasta, että "tiimi ei voi olla, ellei ne ole samassa tilassa"

– Scrum ratkaisuarkkitehti

Väittäisin että se huonontaa sitä. Kasvokkain käytävä kommunikointi on meille lajina tehokkain viestiväline. Oon ihan positiivisesti yllättynyt kuinka hyvin se on toiminu meillä tässä.

– Scrum tuoteomistaja

SAFe-projektista haastateltujen RTE:n, sovellus- ja ratkaisuarkkitehdin mielipiteet tukivat kaikki tätä väitettä. Kaikki olivat samaa mieltä, että yksinkertaisten asioiden hoitaminen etäyhteyksin on helppoa ja tuottavaa, mutta mikäli käsiteltävä asia on teknisempi ja haastavampi, niin etäyhteydet eivät toimi yhtä hyvin kuin fyysinen läsnäolo kokouksessa.

Mun mielestä se [etäpalaveri] ei niinku sovi semmoseen uuden tekemiseen, ideointiin. Se on hyvin huono ajatus. Yritetään piirrellä johonkin Power Pointtiin kaikkien näkyville. Se on vaan ajankäytöllisesti mun mielestä niinku ihan hukkaan heitettyä. Se on niin paljon hitaampaa, kun se että jokainen käy tussilla värittelemässä tai liimailemassa jotain post-it -lappuja ja yritetään hakea jotain ratkasuja. Se on niin paljon tehokkaampaa.

– SAFe-projektin ratkaisuarkkitehti

Scrum-tiimin kehittäjät pitivät hyvänä käytäntönään olevaa tapaa järjestää kerran kuussa yhdestä kahteen päivään mittaisia workshop-tyylisiä päiviä, jolloin kokoontuvat samaan paikkaan käymään läpi haastavampia asioita ohjelmistokehityksessä. Scrum-tiimi myös arveli työntekijöiden pitkän kokemuksen ohjelmistokehittämisestä helpottavan kommunikointia hajautetun tiimin sisällä.

Skypen kautta pystyy hoitamaan pienet ja yksinkertaiset asiat, tai semmoiset asiat, joissa molemmat ovat hyvin selvillä asiasta. Mutta sitten jos siellä on uusia asioita joita kerrotaan, tai henkilöitä joille se tulee ensimmäistä kertaa, niin nenäkkäin on kaikista paras vaihtoehto.

– SAFe-projektin RTE

SAFe-tiimistä haastatellut myös kertoivat, että organisaatiossa on ollut tavoitteena vähentää turhia kokouksia, samalla he kritisoivat organisaation tapaa käyttää SAFe-mallia.

Meillä on ollut tässä syksyn aikana sellainen kehitys liikkeellä, että yritettäisiin vähentää noita palavereita ja koitettaisiin etsiä oikeissa rooleissa olevia oikeita henkilöitä sinne. SAFe:ssa tulee helposti semmoinen malli, että kaikki tietää kaikesta koko ajan kaiken. Ja se ei ole itse tarkoitus.

– SAFe-projektin RTE

Täällä ollaan aivan liikaa palavereissa.

– SAFe-projektin sovellusarkkitehti

Myös Scrum-tiimillä oli haasteita palaverien suuren määrän ja niiden sijoittelun kanssa. Scrum-tiimin haastatelluista toinen kehittäjä piti Skype-palavereja parempina, sillä etäkokouksiin valmistautuminen vie vähemmän aikaa ja niiden aikana voi tehdä muita töitä.

Jos lätkästään päivään kaks Skype-palsua - mikä ei oo mitenkään tavatonta - ja jos siinä on vaikka kaks tai kolme tuntia välissä, niin ei siinä voi mitään koodausjuttuja sinä päivänä enää tehdä. Palsut hyvin nopeasti tuhoaa sitten sen työtehon.

– Scrum sovelluskehittäjä 1

4.3 Yhtäläisyyksiä ja eroja SAFe- ja Scrum-tiimin toimintatavoissa

Scrum-tiimi määritteli aluksi kehittämänsä tuotteen parissa työskenteleviksi henkilöiksi vain tiiminsä jäsenet (tuoteomistaja ja kaksi kehittäjää). Kuitenkin haastattelua jatkettaessa hyvin monelle SAFe:n määrittelemälle roolille löytyi määrätty työntekijä myös heidän organisaatiostaan. Scrum-tiimin organisaatio oli kuitenkin pienempi kuin SAFe-tiimin organisaatio, joten yksi henkilö saattoi vastata useasta SAFe:n määrittelemästä roolista. Scrum-tiimillä ei ollut erillistä liiketoiminnan omistajaa (Business Owner, BO), vaan tuoteomistaja (Product Owner, PO) vastasi sekä BO:n että PO:n roolien vastuista. Tuoteomistaja vastasi omasta mielestään myös osittain Epic Ownerin vastuualueesta yhdessä yrityksen toimitusjohtajan kanssa. Scrum tiimillä ei ollut erikseen Product Manageria tai Portfolio Manageria, vaan budjetoinnista ja visioista vastasi pääsääntöisesti tuoteomistaja ja toimitusjohtaja. Projektin visioinnista oli myös konsultoitu emo-organisaation uusien tuotteiden tuotekehityksestä vastaavia henkilöitä sekä emo-organisaatiossa työskentelevää juristia, joka oli työskennellyt projektin sopimusasioiden parissa. Scrum-tiimistä haastateltu arkkitehti määritteli itsensä ratkaisuarkkitehdiksi ja vastaavansa vain sovelluksen isommista linjauksista. Tarkemman arkkitehtuurin suunnitteluvastuu oli hänen näkemyksensä mukaan ohjelmistokehittäjillä.

No arkkitehtuurista aika paljolti on vastannut kehittäjä ja sitten minä siellä missä on tarvittu sellaisia isompia arkkitehtuurisia linjauksia.

– Scrum ratkaisuarkkitehti

Scrum-tiimin kanssa siis työskenteli paljon henkilöitä rooleissa, joita Scrum-tiimi ei tiedostanut. Scrum-tiimi ei myöskään pitänyt näitä henkilöitä tiimiin kuuluvina jäseninä eikä usein edes jaettuina resursseina. SAFe-tiimissä taas kaikkiin rooleihin oli määrätty yksi henkilö. SAFe-tiimin ratkaisuarkkitehti koki tällaisen toimintatavan jopa tarpeettomaksi.

> *Ainut ongelma mikä tuolla ehkä oli, niin kun joka hatulle laitettiin aina yks henkilö, niin se ei välttämättä vastannu täysin sitä tarvetta. Se et pitääkö esimerkiksi yhdellä tiimillä olla välttämättä aina oma Scrum Master, mikä tekee ainoastaan scrummarin hommia, vaan voisiko Scrum Master olla vaan kiertävä hattu tiimissä, mikä on aika yleinen tapa tehdä sitä.*
>
> *– SAFe ratkaisuarkkitehti*

4.4 Tulkintaeroja tietoturvavaatimuksissa

SAFe-tiimin jäsenet tulkitsivat sekä finanssivalvonnan, Maksukorttiteollisuuden turvallisuusstandardien neuvoston *(Payment Card Industry Security Standards Council, PCI)*, viestintäviraston että organisaation omia tietoturvavaatimuksia eri tavalla kuin Scrum-tiimin jäsenet. Tämä vaikutti osittain tiimien käsitykseen siitä, kuinka nopeasti tiimi kykeni vastaamaan muuttuviin vaatimusmäärittelyihin. Useat SAFe-tiimin jäsenet tulkitsivat PCI DSS:n (Payment Card Industry Data Security Standard) vaatimusten kohdan 6.4.2 *"Separation of duties between development/test and production environments"* [PCIb] sekä sen tarkennuksen *"Practice of dividing steps in a function among different individuals, so as to keep a single individual from being able to subvert the process."* [PCIa] tarkoittavan, että kukaan ohjelmistokehitystiimin ohjelmoiva jäsen ei saa olla minkäänlaisessa yhteydessä tuotantopalvelimen dataan.

Tuotantoon viennissähän on tää PCI DSS, joka sanoo, että on separation of duties, eli siis jotka hakkaa koodia, niin ei muokkaa tuotantoa. Ja ne, jotka muokkaa tuotantoympäristöä, niin ne ei muokkaa koodia. Siellä vois olla ketteröitettävää siinä rajapinnassa ton tuotantoon viennin ja kehityksen välissä.

– SAFe sovelluskehittäjä 2

Yks ongelmahan on kans se regulaatio, että tiimihän ei periaatteessa ite saa tehdä end-to-end kehittämistä, niinkun tuotantoon asti sitä hommaa, koska muuten siellä tulee vaaralliset työyhdistelmät ja muut. Mun mielestä pankkipuolella se ei oo oikeestaan edes sallittuakaan lähtökohtaisesti tehdä niin

– SAFe ratkaisuarkkitehti

Scrum-tiimin jäsenet eivät kuitenkaan koe, että mikään regulaatio kieltäisi ohjelmistokehittäjien pääsyn tuotantoon viennin prosessiin. Scrum-tiimissä luotetaan koodin vertaisarviointiin ja siihen että mahdollisista tietomurroista jäisi varmasti kiinni ja suhteutettuna saavutettuun rikoshyötyyn tällainen toiminta olisi tuottamatonta. Scrum-tiimin ratkaisuarkkitehti kertoo myös käyneensä viestintäviraston kanssa keskustelua regulaation vaatimuksista heidän toimintatapansa kanssa, ja kertoo saaneensa viestintävirastolta päätöksen, jossa kerrotaan ettei regulaatio estä heidän toimintatapaansa.

Ehkä semmoinen semi vainoharhainen asenne meidän mielestä suhtautua hostile intenttiin. Mieluummin mennään semmosella luoton periaatteella. Ja kun luoton pettää, niin se ei tapahdu kuin kerran.

– Scrum ratkaisuarkkitehti

Meillä on niin vahvat auditjäljet kaikesta, että jos tästä nyt lähtis liittoutumaan jotenkin, niin siitä jää kiinni. Et ei sitä voi estää, mutta siitä jää kiinni. Sen takia sitä ei käytännössä kukaan tee.

– Scrum sovelluskehittäjä 1

4.5 Valitun ohjelmistokehitysmallin käytännön eroja

SAFe-mallia käyttänyt tiimi kertoi heillä olevan erikseen Epic Ownerit, sekä Feature-ownerit. Normaalisti SAFe-mallissa vain Epic-tason kehitysaihioilla on erikseen omistajat. Kun Epic -kehitysaihio jaetaan ominaisuuksiksi, niille ei erikseen määrätä omistajaa, vaan kehitysaihion omistaja vastaa myös ominaisuuksien toteutuksen koordinoinnista. Haastateltu sovellusarkkitehti kertoi myös, että jaetun kehitysaihion (Epic) ominaisuuksiakin (Feature) jaetaan vielä pienempiin osiin, joita saattaa lopulta toteuttaa useampi eri tiimi, kun useimmiten kehityksessä ideana on jakaa ominaisuudet niin pieniksi, että yksi tiimi saa toteutettua koko ominaisuuden. SAFe-tiimillä on myös käytössään SAFe:n määrittelemän *Definition of Done* (*DoD*) -käsittelyn lisäksi oma *Definition of Ready* (*DoR*) -käsittely. Näihin käsittelyihin kerättiin emo-organisaation eri tukitiimeiltä kuten tietoturvatiimiltä hyväksyntäleimoja normaalin testauksen lisäksi.

Siellä tuijotettiin DoR:eja ja DoD:eja. Ei päästetty mitään eteenpäin ennen kun ne oli tehty.

– SAFe ratkaisuarkkitehti

Meillä on tietysti Featureissa ja Epiceissä DoR-käsittelyt ja siellä pitää olla tiettyjä lausuntoja tehtynä. Tuntuu että välillä tuli hirveästi nurinaa siitä, että kun on tällaista byrokratiaa, että me joudutaan tekemään DoR-käsittelyä, joudutaan hakemaan leimoja. Mutta mä toisaalta nään sen syyn, että miksi niitä leimoja piti hakea. Siinä on tietysti jotain byrokratiaa osaksi varmaan yhtiön ympäristöistä ja tästä toimialasta johtuvista syistä.

– SAFe sovellusarkkitehti

Scrum-tiimin kehittäjät kertoivat, että heillä vaatimusmäärittelyjä ei jaeta pieniin osiin, vaan tiimi lähtee tekemään kokonaisuuksia ilman kokonaisuuden jakamista pienempiin osiin. Yhden tiketin valmistumiseen meni haastattelujen mukaan yhdestä kahteen kuukautta. Kehittäjät kertoivat myös lähteneensä toteuttamaan vaatimusmäärittelyjä heti ilman tarkempaa suunnitelmaa vaatimusmäärittelyn pienemmistä osista. Scrum-tiimi ei myöskään käyttänyt DoD- tai DoR- määritelmiä, vaan tiimi itse päätti milloin kukin tehtävä on valmis julkaistavaksi. Scrum-tiimin ratkaisuarkkitehti piti hyvänä, ettei tiimille anneta ulkopuolelta DoD- tai DoR-vaatimuksia.

Hyvänä puolena valitsemastaan flow-pohjaista, eli Kanban-tyylisestä ohjelmistokehitysmenetelmästä Scrum-tiimin kehittäjät mainitsivat sen ennalta määriteltyjen seremonioiden vähyyden. Tällöin mallia toteuttavilla henkilöillä ei ole ylimääräisiä mallista johtuvia kokouksia, jotka saattaisivat tuntua turhilta. Tiimi järjesti kuitenkin kerran viikossa statuspalaverin, jossa backlogin järjestystä tarkistettiin. Parannusehdotuksena malliinsa Scrum-tiimin tuoteomistaja ehdotti säännöllisiä retrospektiivejä, joissa voisi konkreettisesti pohtia tiimin mahdollisuuksia parantaa ohjelmistokehitysmenetelmäänsä.

> *Me ollaan sit oikeestaan lähetty tekemään kokonaisuuksia ilman ihan hirveetä pilkkomista. Me ei lähetty tekemään semmosta niinkun jossain Scrummissa nyt periaatteessa kuuluis pilkkoa tosi pieneen. Meillä oli selkeä näkemys missä järjestyksessä me lähdetään tekemään ja sit me vaan lähdettiin tekemään.*
>
> *– Scrum kehittäjä 1*

> *Mitään varsinaista oppikirjajärjestelmäähän ei ollu. Tehdään töitä ja katotaan mitä siitä tulee. Tehdään backlogilta juttuja siinä järjestyksessä, kun ne sattuu olemaan, ja ne tulee valmiiksi kun ne sattuu tulemaan. Ei mitään muuta tarkastelujaksoa, kun se status check kerran viikossa.*
>
> *– Scrum tuoteomistaja*

Pitäis olla säännölliset retrot. Mietittäis miten nimenomaan sitä kehitystä tehtäis paremmin. Ihan konkretiatasolla - ei niin että miten meillä ois täällä kivempaa, vaan miten me nyt devataan kovemmin ja tykimmin.

– Scrum tuoteomistaja

4.6 Valitun ohjelmistokehitysmallin hyötyjä ja haittoja

Kaikki SAFe-tiimistä haastatellut henkilöt kertoivat yhdeksi SAFe:n parhaaksi ominaisuudeksi toimitusjunan ennustettavuuden ja PI-suunnittelupalaverit. Kaikki olivat sitä mieltä, että seuraavan 10 viikon työtehtävien sopiminen on hyvä asia. Samalla myös meneillään olevan organisaatiouudistuksen myötä tulevaa kahden viikon sprinttisykliä pidettiin riskinä. RTE esitti myös kehitysideana pidemmän aikavälin toimintasuunnitelman tekemisen, jolloin olisi helpompi budjetoida kehitystyötä pitkällä aikajänteellä.

Myös SAFe:n hyvin tarkkaa roolitusta pidettiin hyvänä ominaisuutena. Kaikki haastatellut olivat sitä mieltä, että tarkka roolitus edesauttaa uusien henkilöiden perehtymistä projektiin ja ehkäisee työroolien päällekkäisyyksiä, jolloin on helppo tietää kuka vastaa mistäkin asiasta. Myös projektin toiminnasta raportoiminen ulkopuoleisille tahoille nähtiin helppona SAFe:n määrittelemien toimintatapojen edesauttamana. Myös SAFe:n määrittelemiä seremonioita pidettiin toimivina.

Kritiikkiä SAFe:a kohtaan annettiin etenkin organisaation tavasta käyttää SAFe:a. Useimmat haastatellut olivat sitä mieltä, että SAFe-mallista kannattaisi ottaa käyttöön vain organisaation kannalta keskeiset elementit, eikä esimerkiksi täyttää kaikkia SAFe:n määrittelemiä työrooleja, mikäli niiden täyttämisestä ei ole hyötyä. SAFe:n ratkaisuarkkitehti kritisoi, että kaikille SAFe:n rooleille on määrätty yksi henkilö hoitamaan kyseisen roolin tehtävää. Hän ehdotti, että osaa rooleista, kuten Scrum Masterin roolia voisi hoitaa kiertävillä vuoroilla tiimin sisällä.

> *Siit tulee aika raskas hallinnollisesti. Kaiken maailman rooleja joissa on sitten omat tekemiset, ja ne ei välttämättä kaikissa liiketoimintaympäristöissä oo ihan tarpeellisia. Sieltä pitää ottaa ne rusinat pullasta.*
>
> *– SAFe RTE*

Kysyttäessä Scrum-tiimiltä valitun ohjelmistokehitysmallin hyödyistä ja haitoista, he listasivat samoja asioita, kuin mitkä erottivat valitun ohjelmistokehitysmallin perinteisestä Scrum- tai Kanban-mallista. Hyödyllisenä pidettiin sitä, että mitään oppikirjajärjestelmää ei käytetty, vaan käytettiin omiin tarpeisiin soveltuvia seremonioita. Toinen ohjelmistokehittäjistä mainitsi, että jokapäiväisten Daily-seremonioiden pitäminen olisi ollut hänestä turhaa, koska tiimin koko oli niin pieni, että tiimin jäsenet tiesivät ilman seremoniaakin, mitä muut tekivät.

Haitoiksi Scrum-tiimi mainitsi retrospektiivien puutteen. Tiimin jäsenet ehdottivat, että retrospektiivi voitaisiin järjestää Scrum-mallin kahden viikon syklin sijaan kerran kuukaudessa, sillä heidän mukaansa kahdessa viikossa ei ehdi muuttamaan prosesseja tarpeeksi paljon, että voitaisiin arvioida muutoksen hyötyjä.

4.7 Ketterän kehityksen käsityserot

SAFe-tiimistä haastatellut henkilöt kertovat SAFe-mallin hyviksi puoliksi juuri ketteryyden. Erona Scrum-tiimin haastateltuihin, SAFe-tiimin jäsenet pitävät SAFe:n 10 viikon mittaista toimitusjunan sykliä hyvänä ja ketteränä. Scrum-tiimin jäsenet taas väliaikaisesti luopuivat kahden viikon sprinteistä, sillä heidän mielestään ne olivat liian pitkiä, ja näin pitkä sykli heikentää ketterän kehityksen kykyä vastata muuttuviin vaatimuksiin.

Enemmänhän (SAFe:ssa) on kai se idea, että siinä sitoutetaan 10 viikkoo se yksi PI. Niin toisaalta se on ihan hyvä et sen tietää, että okei, nää meidän pitää saada valmiiksi. Mut se, että nyt mennään kahden viikon sprintteihin ja niinkun saattaa olla, että me tehdään sitten sataa asiaa rinnan ja mikään niistä ei valmistu moneen vuoteen jos koko ajan tuutataan, että tehkääpä tämmönenkin.

– SAFe ratkaisuarkkitehti

Mun mielestä on hyvä nähdä pidemmällekin kun vaan toi 10 viikkoa. Kun on tämmöstä innovatiivista kehitystyötä, jolla haetaan uusia markkina-alueita ynnä muita, niin niissä pitää pystyä näkemään pitemmälle tulevaisuuteen.

– SAFe RTE

SAFe RTE:n kommentti ominaisuuksien tekemiseksi valmiiksi ennen niiden tarvitsemista, sekä tällaisten ominaisuuksien vaatimusmäärittelyn muutoksista valmistumisen jälkeen ei myöskään vastaa täysin ketterän kehityksen ideologiaa, jossa ohjelmistoa kehitetään jatkuvasti iteroiden.

Me tehdään palveluita muille, ja me ollaan paljon muiden tekemisistä riippuvaisia. Ja sitten kun me ollaan tehty se palvelu, niin se voi olla että se joka sitä alkaisi hyödyntämään, niin ei olekaan samassa tahdissa meidän kanssa. Ja silloin me ei päästäkään tuotantoon, vaan se jää hyllylle. Ja sitten voi tulla jo, että liiketoiminnalta tulee uusia vaateita siihen samaan versioon.

– SAFe RTE

SAFe-tiimin sovellusarkkitehti kertoi, että heillä kehitysaihiot (Epic) analysoitiin, sekä niihin tehtiin arkkitehtuuri jo ennen kuin ne siirrettiin Programtasolle tuotantoon. Tämä johti siihen, että arkkitehtuuri ja analyysit saattoivat olla jo vuoden vanhoja siinä vaiheessa, kun niitä ruvettiin toteuttamaan. Tästä aiheutui lisätyötä arkkitehtuurin ja analyysien osalta. Tästä huolimatta sovellusarkkitehti piti etukäteen tehtyjä analyyseja hyvänä asiana.

> *Epicien analyysissä saattoi olla kaikki arkkitehtuurijutut tehtynä vuotta aikaisemmin. Monesti ne jouduttiin sitten kattomaan uudestaan ja tekemään pientä stilisointia, että mitä siinä oikeasti on tarkoitus toteuttaa ja sitten jossain kohtaa huomattiin, että joku toinen tiimin saattoi ollut jo tehnyt sen.*
>
> *– SAFe sovellusarkkitehti*

5 Pohdinta

Internetin blogeista löytyy paljon SAFe:a kritisoivia kirjoituksia. Vaikka kritiikkiä SAFe:a kohtaan löytyy paljon internetistä, sitä kritisoivia tieteellisiä tutkimuksia oli erittäin hankala löytää. Eräs selitys kriittisten tutkimusten vähyydelle voi olla, että ohjelmistokehitysmenetelmistä konsultointi ja koulutukset ovat maailmanlaajuisesti isoa liiketoimintaa, ja SAFe:n takana oleva Scaled Agile Inc. on rahoittanut paljon positiivista tutkimusta SAFe:n hyödyistä.

Haastattelujen perusteella SAFe- ja Scrum-tiimien näkemykset ketterästä kehityksestä eroavat paljon toisistaan. SAFe-tiimin mielestä 10 viikon sykli on ketterää kehitystä, kun taas Scrum-tiimin mielestä kahden viikon sykli on liian pitkä ollakseen ketterä. Yleisesti ketterässä kehityksessä käytetty kahden viikon sykli on osoittautunut toimivaksi useimmissa organisaatioissa. Kaksi viikkoa on suhteellisen lyhyt aika, mutta siihen voidaan suunnitella kuitenkin loogisesti ja jäsennellysti työtehtäviä. Hyvänä jatkotutkimusaiheena olisi tutkia eripituisten sprittisyklien vaikutusta työtehokkuuteen sekä tiimin toiminnan ennustettavuuteen. Hyödyttääkö alle kahden viikon sprinttisykli vai tekeekö tiimi tällaisessa tilanteessa enemmän turhaa työtä? Voiko liian lyhyt sprinttisykli vaikuttaa tiimin toimintaan siten, että päämäärätietoinen ohjelmiston kehittäminen kärsii?

SAFe-tiimin ratkaisuarkkitehti ilmaisi haastattelussa epäilyksensä kahden viikon sprinttisykliin siirtymisen toimivuudesta. Hänen mukaansa tiimi tekisi silloin liian montaa työtehtävää samanaikaisesti, eikä mitään saataisi valmiiksi. Sprinttisyklin lyhentäminen vaatii oletettavasti erilaista tiimin työnteon koordinointia, kuin pidemmän sprinttisyklin koordinointi. Sprinttisyklin pituuden ei kuitenkaan kuuluisi vaikuttaa suoraan tiimin samanaikaisesti tehtävien töiden määrään. Tästä hyvänä esimerkkinä on Kanban-malli, joka ei toimi syklisenä sprinttimallina, mutta rajoittaa kuitenkin samanaikaisesti suoritettavien työtehtävien määrää.

Tutkimuksessa mukana ollut Scrum-tiimi ei noudattanut Scrumin ja Kanbanin periaatetta, jossa toteutettavat ominaisuudet jaetaan pieniin osiin. Scrum-tiimin kehittäjät kertoivat, etteivät olleet jakaneet kehityksen aikana vaatimusmäärittelyä Scrumin mukaisiin pieniin tehtäviin (task), vaan olivat suoraan tehneet yhdestä kahteen kuukauden mittaisia tehtäviä. Näin pitkien tehtävien tekeminen jakamatta niitä pienempiin osiin vaikeuttaa oletettavasti tiimin toiminnan edistymisen seuraamista sekä yksittäisten vaativien toiminnallisuuksien tekemisen ennakoitavuutta.

Haastattelujen perusteella Scrum-tiimin ja SAFe-tiimin käsitykset omien tiimiensä koosta ja rakenteesta erosivat huomattavasti toisistaan. Scrum-tiimi määritteli aluksi tiimiinsä kuuluvan vain tuoteomistajan ja kaksi kehittäjää, SAFe-tiimi puolestaan määritteli tiimiinsä kuuluvan kaikki SAFe-ohjelmistokehityskehyksen määrittelemien roolien työntekijät. Monelle SAFe:n määrittelemälle roolille löytyi kuitenkin vastuuhenkilö Scrum-tiimin kanssa yhteistyötä tekevistä tahoista. Scrum-tiimin kanssa siis työskenteli paljon henkilöitä tunnistamattomissa rooleissa, eikä Scrum-tiimi pitänyt näitä henkilöitä tiimiinsä kuuluvina jäseninä. Tiimi ei maininnut aluksi näitä henkilöitä edes jaettuina resursseina. Scrum-tiimi ei siis tiedostanut näiden työroolien kuuluvan Scrum-tiimin työmäärään. Tällainen toiminta saattaa aiheuttaa väärinkäsityksiä tiimin tehokkuudesta työntekijämäärään suhteutettuna, mikäli tällaisen tiimin työtä verrataan henkilömäärältään suurempaan tiimiin, jossa kaikkien työroolien työpanos lasketaan mukaan.

SAFe-tiimin ratkaisuarkkitehti kritisoi toimintamallia, jossa jokaiselle roolille on määrätty yksi työntekijä. Hänen mukaansa joitain rooleja voitaisiin hoitaa myös kiertävällä vastuulla. Tämä toimintamalli oli osittain käytössä Scrum-tiimin organisaatiossa, jossa yksi henkilö vastasi useiden roolien työtehtävistä. Työntekijämäärän kasvaessa kuitenkin oletettavasti monen roolin työtehtävien määrä lisääntyy, jolloin yhden henkilön työpanoksella ei välttämättä pysty hoitamaan usean roolin vaatimuksia. Hyvänä jatkotutkimusaiheena olisikin tutkia työroolien työmäärän kasvua suhteutettuna tiimin henkilöstömäärään.

Tärkeimmäksi muutokseksi perinteisestä ohjelmistokehityksestä ketterään menetelmään siirtymisessä oli Misra et al. tutkimuksessa mainittu siirtyminen käskytä-ja-kontrolloi -toimintamallista ketterän kehityksen mukaiseen opasta-ja-osallistu -toimintamalliin. Tällaisessa mallissa myös toteuttava taho osallistuu päätöksentekoon [Sub10]. SAFe-tiimin kokemat organisaation vanhoihin toimintatapoihin liittyvät ongelmat, joissa organisaation päätöksentekoja tekevä osasto ei ole sitoutunut käyttämään SAFe-mallia, vaikuttavat olevan samoja, kuin Kalendan, Misran et al. ja Collin et al. tutkimuksissaan havaitsemat ongelmat. Collin et al. olivat havainneet RAY:lla, että mikäli kaikki organisaation osastot eivät ota SAFe-ohjelmistokehityskehystä käyttöön, niin ketterää ohjelmistokehitystä ei saada toimimaan [Col15]. Kalenda puolestaan havaitsi pro gradu -tutkielmassaan, että organisaation työntekijöiden muutosvastarinta on yleinen haaste ketterien menetelmien käyttöön ottamisessa [Kal17]. Myös Putta et al. ovat tehneet vastaavat havainnot SAFe-mallin käyttöön otossa. Heidän tutkimuksensa mukaan suurin ongelma SAFe-mallin käyttöön ottamisessa on muutosvastarinta [PPL18]. Pancholi ja Grover esittävät väitteen, että SAFe-mallia markkinoidaan ensisijaisesti organisaatioille, jotka pelkäävät muutosta, mutta haluavat kuitenkin lisätä tuottavuutta ja vähentää virheiden määrää. Heidän mukaansa SAFe on kehitetty johtajille, jotka eivät ymmärrä ketterää kehitystä [PG14]. Tätä väitettä tukee myös SAFe-tiimin kokemukset organisaation vanhojen rakenteiden olemassaolosta, vaikka käytössä onkin teoreettisesti SAFe-malli.

Tutkimuksessani havaitsin, että SAFe-organisaation toimintaa ideasta tuotantoon ei voida pitää kovin nopeasti muuttuviin vaatimuksiin vastaavana. Nopeimmillaankin ideasta tehty kehitysaihio (Epic) on yhden PI-jakson, eli 10 viikkoa suunnittelussa, jonka jälkeen seuraavan PI-jakson toteutuksessa. Toteutuksen jälkeen uusi toiminnallisuus päätyy vielä yhden PI-jakson ajaksi testaukseen ja tuotantoon vientiin. Tällöin siis yhden idean tuotantoon saattaminen kestää lyhimmilläänkin 30 viikkoa. Tämä tosin johtuu suurimmaksi osaksi organisaation toimintamalleista sekä regulaatiovaatimuksista, eikä johdu suoranaisesti SAFe-mallin käytöstä. Haastattelujen perusteella tätä sykliä olisi mahdollista myös nopeuttaa, mutta haastatellut henkilöt eivät esittäneet

lainkaan konkreettisia muutosehdotuksia syklin nopeuttamiseksi.

Misra et al. tekemän tutkimuksen mukaan ketteriin menetelmiin siirryttäessä projektien kokoa tulee pienentää, projekteissa täytyy keskittyä paljon prototyyppien tekemiseen, jatkuvaan testaamiseen, viikoittaisiin julkaisuihin ja jatkuvaan palautteen vastaanottamiseen. Näihin tavoitteisiin päästään toteuttamalla projektia pienin askelein eteenpäin [Sub10]. SAFe-mallia käyttävän tiimin usean kymmenen viikon mittainen julkaisusykli sekä keskitetyt testaus- ja julkaisuputket eivät näin ollen täytä Misran et al. vaatimuksia ketterään kehittämiseen siirtymisen muutoksista.

SAFe-tiimin RTE:n näkemys ongelmista uusien ominaisuuksien toimintavaatimusten muutoksista toteutuksen jälkeen ei vaikuta ketterän kehityksen filosofian mukaiselta, vaan pikemminkin vesiputousmallin toimintaperiaatteelta. Vaikka samaan ominaisuuteen tehtävät iteraatiosyklit oletettavasti tuntuvatkin turhauttavilta, kaikkien uusien ominaisuuksien jäädyttäminen tuotantoon viennin jälkeen tekisi ohjelmiston jatkokehittämisestä mahdotonta.

Tiimien tulkintaerot tietoturvavaatimuksista voivat osittain johtua tiimien eri sijoittelusta organisaation rakenteessa. SAFe-tiimi työskenteli suoraan emo-organisaation brändin alaisuudessa ja Scrum-tiimillä oli oma osakeyhtiönsä erillisellä nimellä. Oletettavasti emo-organisaation suuren ja arvokkaan brändin alaisuudessa työskentelevät tiimit joutuvat suhtautumaan huomattavasti vakavammin brändiarvoon mahdollisesti vaikuttaviin asioihin, jolloin kaikki vaatimukset ja standardit tulkitaan jopa ylivarovaisesti. Kun kehitykseen tuodaan mukaan paljon ylimääräistä byrokratiaa, se heikentää kehitystiimin mahdollisuuksia reagoida nopeasti muuttuviin vaatimuksiin. Tämä taas johtaa siihen, että oikeasti ketterää kehittämistä, joka vastaisi välittömästi havaittuihin muutostarpeisiin, on vaikeaa tehdä.

Misra et al. mainitsevat tutkimuksessaan, että eräs keskeinen filosofinen ero ketterän kehityksen mallien ja perinteisten mallien välillä on, että ketterässä kehityksessä pyritään pois dokumentaatiokeskeisestä ohjelmistokehityksestä, ja

pyritään vahvistamaan *toimiva tuote* -keskeistä ohjelmistokehitysmallia [Sub10]. Useissa tapauksissa tämä varmasti lyhentää aikaa tuotantoon pääsylle, mutta tarkasti reguloiduissa projekteissa tarkan dokumentaation tekeminen on kuitenkin pakollista. Siitä huolimatta tässä tutkimuksessa vertaillut samat regulaatiovaatimukset täyttävät projektit käyttivät haastattelujen perusteella dokumentointiin ja ominaisuuksien hyväksyntätestaamiseen hyvin erilaisen määrän aikaa. Oletettavasti SAFe-mallia käyttävän tiimin hyväksyntätestausprosessit olivat siis jopa liian raskaita, ja osaltaan heikensivät tiimin kykyä tuottaa ohjelmistoa ketterästi.

Haastattelujen perusteella molemmat tiimit kokivat hajautetun tiimin toiminnan ongelmalliseksi. SAFe-tiimillä oli suuria vaikeuksia eri maassa toimivan tiimin työn seuraamisen ja kommunikoinnin suhteen. Scrum-tiimi oli aluksi pelännyt hajautetun tiimin vaikuttavan heikentävästi kommunikointiin ja työtehoon. Haastatelluista lähes kaikki olivat sitä mieltä, että etenkin vaativampien asioiden läpikäynti on helpompaa kokouksessa, jossa kaikki osallistujat ovat fyysisesti läsnä. Sekä Leffingwell että Simons ovat havainneet, että projekti- ja prosessispesifisen tiedon jakaminen on ongelmallista erillään työskentelevien tiimien keskuudessa [Lef07] [Sim06]. Myös ketterän ohjelmistokehityksen julistus (Agile manifesto) kehottaa tiimejä jakamaan tietoja kasvokkain käytävin keskusteluin. Tiettyihin aihealueisiin keskittyviä tiimejä kannattaisi keskittää samaan toimistoon, jossa olisi helppo keskustella kasvokkain. Myös projektien jakamista pienempiin osaprojekteihin voisi harkita ratkaisuksi. Tällöin oletettavasti kommunikointitarve tiimien välillä vähenisi. Välttämättömät projektien väliset keskustelut voitaisiin järjestää siten, että kaikki keskusteluissa tarvittavat henkilöt olisivat fyysisesti läsnä.

Kaikilla haastatelluilla oli yhtenäinen näkemys siitä, että koko organisaation laajuisesti järjestetään liikaa palavereja. Haastatteluissa tuli myös esille, että mikäli saman työpäivän aikana on useampi palaveri eri aikoihin päivästä, ei niiden välissä ehdi organisoitumaan muuhun työhön. Hyvänä jatkotutkimuksena voitaisiin tutkia tiimien työtehokkuutta nykyisin järjestelyin, sekä siirtäen palaverit olemaan peräkkäin. Myös osan palavereista siirtämistä sähköpostiviestittelyksi, sekä tämän vaikutuksesta työtehokkuuteen voitaisiin tutkia.

Scrum-tiimistä haastateltu kehittäjä piti etäpalavereja parempina kuin kokoushuoneessa järjestettyä palaveria, sillä etäpalaverin aikana pystyi tekemään muita töitä samaan aikaan. Oletettavasti etäpalaverin aikana muiden töiden tekeminen kuitenkin häiritsee palaveriin osallistumista sekä samaan aikaan muun työn ohella palaveriin keskittyminen heikentää työn tuottavuutta. Mikäli tällainen käytös on yleistä organisaatiossa, voitaisiin pohtia ovatko muita töitä etäpalaverien aikaan tekevät henkilöt oleellisia henkilöitä palaverin päämäärään pääsemisen suhteen, vai voisiko heidät jättää pois palaverista.

Haastattelujakson aikana alkanut organisaatiouudistus näkyi useissa haastatteluvastauksissa epäilyksenä uuden tulevan mallin toimivuudesta. Tämän tutkimuksen aikana organisaatiouudistusta ei kuitenkaan saatu täysin toteutettua, eikä sen tuomista muutoksista ole vielä näyttöä.

Tämän sekä muiden tässä tutkimuksessa käsiteltyjen tutkimusten perusteella SAFe-ohjelmistokehityskehys on yleisesti hyvin monimutkainen ja vaikeasti hallittava. Suuret projektit olisi hyvä jakaa pienemmiksi osaprojekteiksi, jolloin SAFe:n kaltaista monimutkaista ohjelmistokehityskehystä ei välttämättä tarvittaisi. SAFe-tiimin käyttämä 30 viikon mittainen sykli ideasta tuotantoon ei ole nopeasti muuttuviin vaatimusmäärittelyihin vastaavaa toimintaa, ja ehdottaisin tämän syklin lyhentämistä ja yksinkertaistamista. Tutkimukseni perusteella vaatimusmäärittelyt kannattaa jakaa pieniin osiin, sillä tällöin työnteon ennustettavuus paranee. Pidän Scrum-tiimin käyttämää tekniikkaa, jossa vaatimusmäärittelyä ei jaeta lainkaan pienempiin osiin, huonosti ennustettavana.

Tutkimukseni, sekä muiden tässä tutkimuksessa käsiteltyjen tutkimusten perusteella työntekijät kannattaa sijoittaa samaan fyysiseen sijaintiin, jolloin työntekijöiden välinen kommunikointi paranee. Palaverien määrää tulisi tutkimukseni perusteella vähentää. Osan palavereista voisi siirtää sähköpostiviesteiksi. Myös palaverien tarkempaa sijoittelua työpäivään alkuun tai loppuun tulisi harkita, sillä tutkimukseni perusteella palaverit katkaisevat työntekijöiden työnteon ja häiritsevät keskittymistä vaativaa työtä.

5.1 Tutkimuksen luotettavuus

Jokaisessa yksittäisessä tutkimuksessa tulee arvioida tehdyn tutkimuksen luotettavuutta [Tuo18]. Laadullisessa tutkimuksessa tulee arvioida sekä tutkimuksen totuutta että tutkijan objektiivisuutta [Tuo18][RH08]. Tarkasteltaessa tutkijan objektiivisuutta, täytyy se jakaa havaintojen luotettavuuteen ja havaintojen puolueettomuuteen. Puolueettomuutta tarkastellessa on huomioitava, vaikuttaako tutkijan oma persoona vastausten analyysiin. Laadullisessa tutkimuksessa on myönnettävä, että tutkijan persoona vaikuttaa vastausten analyysiin, sillä tutkija on tutkimusasetelman luoja ja tulkitsija. Myös luotettavuutta analysoitaessa on pyrittävä huomioimaan tutkijan puolueettomuusnäkökulma [Tuo18].

Laadullisen tutkimuksen pätevyyttä voidaan arvioida neljällä näkökulmalla: rakenteellisella pätevyydellä, sisäisellä pätevyydellä, ulkoisella pätevyydellä sekä luotettavuudella. Rakenteellisella pätevyydellä tarkoitetaan sitä, että tutkija on tutkinut oikeasti sitä aihetta, jota varten tutkimuskysymykset on luotu. Sisäisessä pätevyydessä tutkijan tulee ottaa huomioon syy-yhteyksiä tutkiessa, että mitkä kaikki tekijät vaikuttavat tutkittavaan aiheeseen. Ulkoisessa pätevyydessä tutkijan on huomioitava, missä määrin tulokset ovat yleistettävissä ja mitkä havainnot kiinnostavat muita. On huomioitava, että tapaustutkimuksen otanta ei ole yleensä tilastollisesti merkittävä, mutta joistakin tutkimustuloksista voidaan löytää yhteisiä piirteitä, jotka ovat yleistettävissä. Luotettavuutta arvioitaessa on huomioitava, missä määrin kerätty tieto ja analyysi ovat riippuvaisia tutkijoista, ja mikäli joku toinen tekisi samanlaisen tutkimuksen, olisivatko tulokset samoja [RH08].

Tässä tutkimuksessa rakenteellista pätevyyttä voidaan pitää hyvänä, sillä puolistrukturoidun teemahaastattelun kysymykset luotiin juuri tätä tutkimusta varten. Sisäistä pätevyyttä on pyritty pitämään korkealla muun muassa ottamalla huomioon haastateltujen henkilöiden työroolien vaikutukset heidän näkökantaansa tutkittavasta aiheesta. Ulkoisessa pätevyydessä tulee huomioida vastaajien pieni määrä (yhdeksän henkilöä). Tutkimusvastauksista on kuitenkin pyritty löytämään yhtäläisyyksiä muiden tutkimusten tuloksiin, ja näiden

pohjalta löytämään yleistettävissä olevia tuloksia. Tutkimuksen luotettavuudessa on pyritty objektiivisuuteen muun muassa sillä, että tutkijana en ole työskennellyt tutkimuskohteiden kanssa aiemmin.

6 Yhteenveto

Tutkielmassa selvitettiin puolistrukturoidun teemahaastattelun pohjalta tehdyn kvalitatiivisen tutkimuksen avulla kahden eri ohjelmistokehitystiimin ketterän kehityksen toimintaa suomalaisessa finanssialan yrityksessä. Toinen tiimeistä käytti ohjelmistokehityksessään Scaled Agile Framework (SAFe) -ohjelmistokehityskehystä ja toinen tiimi käytti Scrum-mallia. Tiimit tuottivat keskenään kilpailevia henkilön vahvan tunnistamisen palveluita. Tutkielmassa vertailtiin tiimien toimintatapoja keskenään sekä tiimien toimintatapoja suhteutettuna ohjelmistokehitysmalleihin, joita tiimit kertoivat käyttävänsä. Tutkielmassa selvitettiin myös tiimien käsityseroja ketterästä kehittämisestä, sekä tiimien kokemia haasteita ketterästä kehittämisestä. Näitä haasteita verrattiin aiemmissa tutkimuksissa ilmi tulleisiin haasteisiin. Tutkielmassa tehdyt havainnot antavat mahdollisuuden tiimin keskinäisen kommunikaation parantamiseen sekä avaavat näkökulmia siihen, kuinka tarkasti SAFe:n tai Scrumin käytäntöjä noudatetaan ohjelmistokehityksessä. Tutkimusta varten haastateltiin yhdeksää henkilöä useista eri työrooleista, siten että SAFe ohjelmistokehityskehystä käyttävästä tiimistä haastateltiin viittä henkilöä ja Scrum-mallia käyttävästä tiimistä haastateltiin neljää henkilöä.

Haastattelujen perusteella tiimeillä oli hyvin erilaiset käsitykset ketterän kehityksen periaatteesta, että muuttuviin vaatimuksiin reagoidaan nopeasti. SAFe-tiimin jäsenet olivat sitä mieltä, että 10 viikon sykli vastaa tarpeeksi nopeasti muuttuviin vaatimuksiin, kun Scrum-tiimin mielestä kahden viikon syklillä ei pystytä vastaamaan tarpeeksi nopeasti muuttuviin vaatimuksiin. Kuitenkin SAFe-tiimissä muut kuin kehittäjät kertoivat, että käytetty ohjelmistokehityskehys ei heidän mukaansa vaikuta ketterältä kuin alimmalla ohjelmistokehitys tasolla, sillä ylemmillä tasoilla käsiteltävien kehitysaihioiden (Epic) valmistuminen kestää puoli vuotta ja sen parissa saattaa työskennellä jopa kaksi tiimiä. Puolen vuoden mittainen kehitysaihio (Epic) ei heidän mielestään ollut tarpeeksi joustava, eikä siis täyttänyt ketterän kehityksen vaatimuksia nopeasti muuttuviin vaatimusmäärittelyihin vastaamisesta.

Tiimeillä oli myös erilainen kokoonpano sekä käsitys tiimeihin kuuluvista jäsenistä. SAFe-tiimillä jokaiselle SAFe:n määrittelemälle roolille oli määrätty yksi työntekijä, kun Scrum-tiimin organisaatiossa yksi henkilö saattoi vastata useasta eri työroolista. Scrum-tiimi ei myöskään pitänyt tiiminsä jäseninä kaikkia sellaisista rooleista vastanneita henkilöitä, joita SAFe-tiimi piti tiiminsä jäseninä. SAFe-tiimi kuitenkin kritisoi heidän omaa ratkaisuaan, jossa jokaiselle roolille on määrätty yksittäinen henkilö, koska heidän mukaansa joitain rooleja olisi voitu hoitaa yhdessä toisen roolin kanssa.

SAFe-tiimin organisaatiossa oli myös haastattelujen perusteella havaittavissa muutosvastarintaa SAFe-ohjelmistokehityskehykseen siirtymisen kanssa. Osa organisaatiosta ei ollut siirtynyt käyttämään SAFe:a, vaan oli jatkanut toimintaansa vanhojen mallien mukaan. Tämä heikensi SAFe-ohjelmistokehityskehyksen toimintaa. Samanlaisia havaintoja on tehty useissa muissakin tutkimuksissa, kuten Kalendan, Mistan et al. ja Collin et al. tutkimuksissa [Kal17] [Sub10] [Col15].

Tutkimuksessa mukana olleet tiimit tulkitsivat regulaatio ja tietoturvavaatimuksia eri tavalla. Scrum-tiimi pyrki pitämään kehityksen mahdollisimman joustavana tulkitsemalla vaatimuksia löyhemmin kuin SAFe-tiimi, joka tulkitsi kaikki vaatimukset hyvin tiukasti. Molemmat tiimit kuitenkin toteuttivat samat vaatimukset, joten voidaan olettaa SAFe-tiimin tulkinneen vaatimuksia liiankin tarkasti. Sekä regulaatiovaatimukset, että organisaation tietoturvavaatimuksetkin vaativat tiimeillä olevan ohjelmistostaan dokumentaatio. Scrum-tiimi piti dokumentaation hyvin minimaalisena, kun SAFe-tiimi taas dokumentoi asiat erittäin tarkasti. Vaikka ketterän kehityksen filosofian mukaista on keskittyä ohjelmistoon dokumentaation sijaan [Sub10], niin oletettavasti uutena työntekijänä ohjelmistoon tutustuminen on helpompaa, mikäli dokumentaatio on kattava.

Kaikilla haastatelluilla oli yhteinen näkemys siitä, että organisaatio järjestää liikaa palavereja, ja tämä häiritsee muun työn tekoa. Tämä ongelma oli organisaatiossa myös haastattelujen perusteella havainnoitu, ja palaverien määrää pyritään tulevaisuudessa vähentämään. Suurin osa haastatelluista piti etäpalavereja parempana ratkaisuna etenkin yksinkertaisia asioita hoidettaessa. Tutkimukseni perusteella palaverien ajankohtia tulisi miettiä tarkemmin, jolloin ne mahdollisesti saataisiin työpäivän alkuun tai loppuun, jolloin työntekijälle jäisi mahdollisimman pitkä keskeytyksetön aika hoitaa muita työtehtäviään. Osa palavereista voitaisiin myös muuttaa sähköpostiviestittelyksi. Tutkimukseni perusteella suuret projektit tulisi jakaa pienemmiksi osaprojekteiksi, jolloin projektien hallinta olisi helpompaa ja ketterän kehityksen vaatimuksiin muuttuviin vaatimusmäärittelyihin nopeasti vastaamisesta olisi helpompaa.

Lähteet

[AGW08] Abbas, Noura, Gravell, Andrew M. ja Wills, Gary B.: *Historical Roots of Agile Methods: Where Did "Agile Thinking" Come From?* Teoksessa Abrahamsson, Pekka, Baskerville, Richard, Conboy, Kieran, Fitzgerald, Brian, Morgan, Lorraine ja Wang, Xiaofeng (toimittajat): *Agile Processes in Software Engineering and Extreme Programming*, sivut 94–103, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[AMO13] Ahmad, M. O., Markkula, J. ja Oivo, M.: *Kanban in software development: A systematic literature review.* Teoksessa *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, sivut 9–16, Sep. 2013.

[BBvB+] Beck, Kent, Beedle, Mike, Bennekum, Arie van, Cockburn, Alistair, Cunningham, Ward, Fowler, Martin, Grenning, James, Highsmith, Jim, Hunt, Andrew, Jeffries, Ron, Kern, Jon, Marick, Brian, Martin, Robert C., Mellor, Steve, Schwaber, Ken, Sutherland, Jeff ja Thomas, Dave: *Julistuksen takana olevat periaatteet.* `https://agilemanifesto.org/iso/fi/principles.html`, vierailtu 04.04.2019 .

[Ben83] Benington, H. D.: *Production of Large Computer Programs.* Annals of the History of Computing, 5(4):350–361, Oct 1983.

[Col15] Collin, Jari (ed.); Hiekkanen, Kari (ed.); Korhonen Janne J. (ed.); Halén Marco (ed.); Itälä Timo (ed.); Helenius Mika (ed.): *IT Leadership in Transition - The Impact of Digitalization on Finnish Organizations.* D4 Julkaistu kehittämis- tai tutkimusraportti tai -selvitys, 2015. `http://urn.fi/URN:ISBN:978-952-60-6243-3`.

[DD08] Dybå, Tore ja Dingsøyr, Torgeir: *Empirical studies of agile software development: A systematic review.* Information and Software Technology, 50(9):833 – 859, 2008, ISSN 0950-5849. `http://www.sciencedirect.com/science/article/pii/S0950584908000256`.

[ELS05] Erickson, John, Lyytinen, Kalle ja Siau, Keng: *Agile Modeling, Agile Software Development, and Extreme Programming: The State of Research*. 16(4):88–100, 2005, ISSN 1063-8016. http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/jdm.2005100105.

[EP17] Ebert, C. ja Paasivaara, M.: *Scaling Agile*. IEEE Software, 34(6):98–103, November 2017, ISSN 0740-7459.

[Hir08] Hirsjärvi, Sirkka, kirjoittaja: *Tutkimushaastattelu : teemahaastattelun teoria ja käytäntö*. Gaudeamus Helsinki University Press, Helsinki, 2008. Lisäpainokset: [Lisäp.] 2009. - 2010. - 2011.

[Ins] Instituten, Project Management: *Success in Disruptive Times: Expanding the Value Delivery Landscape to Address the High Cost of Low Performance*. https://www.pmi.org/-/media/pmi/documents/public/pdf/learning/thought-leadership/pulse/pulse-of-the-profession-2018.pdf, vierailtu 16.10.2019 .

[Kal17] Kalenda, Martin: *"Scaling Agile Software Development in Large Organizations*. master's thesis, Faculty of Informatics, Masaryk Univ., 2017. is.muni.cz/th/410499/fi_m/masters_thesis.pdf.

[KS] Ken Schwaber, Jeff Sutherland: *The Scrum Guide*. https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pd, vierailtu 11.07.2019 .

[LB03] Larman, C. ja Basili, V. R.: *Iterative and incremental developments. a brief history*. Computer, 36(6):47–56, June 2003, ISSN 0018-9162.

[Lef07] Leffingwell, Dean: *Scaling Software Agility: Best Practices for Large Enterprises*. Addison-Wesley Professional, 2007.

[MW] Mike West, William Holz: *Result Summary Agile in the Enterprise*. https://circle.gartner.com/portals/2/resources/pdf/agile%20in%20the%20enterprise%202017.pdf, vierailtu 16.10.2019 .

[NB07] Nerur, Sridhar ja Balijepally, VenuGopal: *Theoretical Reflections on Agile Development Methodologies.* Commun. ACM, 50(3):79–83, maaliskuu 2007, ISSN 0001-0782. https://doi.org/10.1145/1226736.1226739.

[Ove] Overflow, Stack: *Developer Survey Results 2018.* https://insights.stackoverflow.com/survey/2018#development-practices, vierailtu 16.10.2019 .

[Pay] Payravi, Kevin: *File:Scrum diagram (labelled).png.* https://commons.wikimedia.org/wiki/File:Scrum_diagram_(labelled).png, vierailtu 05.07.2018 .

[PC12] Poppendieck, M. ja Cusumano, M. A.: *Lean Software Development: A Tutorial.* IEEE Software, 29(5):26–32, Sep. 2012, ISSN 0740-7459.

[PCIa] PCI, Security Standards Council: *Payment Card Industry (PCI) Data Security Standard (DSS) and Payment Application Data Security Standard (PA-DSS).* https://www.pcisecuritystandards.org/documents/PCI_DSS_Glossary_v3-2.pdf, vierailtu 08.01.2019 .

[PCIb] PCI, Security Standards Council: *Payment Card Industry (PCI) Data Security Standard, Requirements and Security Assessment Procedures.* https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2-1.pdf, vierailtu 07.07.2019 .

[PG14] Pancholi, Aditya ja Grover, Sapna: *Scaled Agile Framework: A Blight.* 2014.

[PLH12] Paasivaara, M., Lassenius, C. ja Heikkilä, V. T.: *Inter-team coordination in large-scale globally distributed scrum: Do Scrum-of-Scrums really work?* Teoksessa *Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, sivut 235–238, Sept 2012.

[Pop07] Poppendieck, M.: *Lean Software Development.* Teoksessa *29th International Conference on Software Engineering (ICSE'07 Companion)*, sivut 165–166, May 2007.

[PPL18] Putta, Abheeshta, Paasivaara, Maria ja Lassenius, Casper: *Benefits and Challenges of Adopting the Scaled Agile Framework (SAFe): Preliminary Results from a Multivocal Literature Review*. Teoksessa Kuhrmann, Marco, Schneider, Kurt, Pfahl, Dietmar, Amasaki, Sousuke, Ciolkowski, Marcus, Hebig, Regina, Tell, Paolo, Klünder, Jil ja Küpper, Steffen (toimittajat): *Product-Focused Software Process Improvement*, sivut 334–351, Cham, 2018. Springer International Publishing.

[Raz16] Razzak, M. A.: *An Empirical Study on Lean and Agile Methods in Global Software Development*. Teoksessa *2016 IEEE 11th International Conference on Global Software Engineering Workshops (ICGSEW)*, sivut 61–64, Aug 2016.

[RH08] Runeson, Per ja Höst, Martin: *Guidelines for conducting and reporting case study research in software engineering*. Empirical Software Engineering, 14(2):131, 2008, ISSN 1573-7616. `https://doi.org/10.1007/s10664-008-9102-8`.

[Roy87] Royce, W. W.: *Managing the Development of Large Software Systems: Concepts and Techniques*. Teoksessa *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87, sivu 328–338, Washington, DC, USA, 1987. IEEE Computer Society Press.

[SAa] Scaled Agile, Inc: *Agile Release Train - Scaled Agile Framework*. `https://v45.scaledagileframework.com/agile-release-train/`, vierailtu 18.04.2019 .

[SAb] Scaled Agile, Inc: *Agile Teams - Scaled Agile Framework*. `https://v45.scaledagileframework.com/agile-teams/`, vierailtu 16.04.2019 .

[SAc] Scaled Agile, Inc: *Enterprise Architect - Scaled Agile Framework*. `https://v45.scaledagileframework.com/enterprise-architect/`, vierailtu 18.07.2019 .

[SAd] Scaled Agile, Inc: *Epic Owner - Scaled Agile Framework*. `https://v45.scaledagileframework.com/epic-owner/`, vierailtu 07.04.2019 .

[SAe] Scaled Agile, Inc: *Large Solution Level - Scaled Agile Framework*. https://v45.scaledagileframework.com/large-solution-level/, vierailtu 08.11.2019 .

[SAf] Scaled Agile, Inc: *Lean Portfolio Management - Scaled Agile Framework*. https://v45.scaledagileframework.com/lean-portfolio-management/, vierailtu 20.11.2019 .

[SAg] Scaled Agile, Inc: *PI Planning - Scaled Agile Framework*. https://v45.scaledagileframework.com/pi-planning/, vierailtu 19.12.2019 .

[SAh] Scaled Agile, Inc: *Program Increment - Scaled Agile Framework*. https://v45.scaledagileframework.com/program-increment/, vierailtu 18.04.2019 .

[SAi] Scaled Agile, Inc: *Release Train Engineer and Solution Train - Scaled Agile Framework*. https://www.scaledagileframework.com/release-train-engineer-and-solution-train-engineer/, vierailtu 24.07.2019 .

[SAj] Scaled Agile, Inc: *SAFe website*. https://www.scaledagileframework.com/, vierailtu 05.07.2018 .

[SAk] Scaled Agile, Inc: *System and Solution Architect/Engineering - Scaled Agile Framework*. https://v45.scaledagileframework.com/system-and-solution-architect-engineering/, vierailtu 18.07.2019 .

[SAl] Scaled Agile, Inc: *What Is SAFe*. https://www.scaledagileframework.com/what-is-safe, vierailtu 05.07.2018 .

[SBS17] Srivastava, A., Bhardwaj, S. ja Saraswat, S.: *SCRUM model for agile methodology*. Teoksessa *2017 International Conference on Computing, Communication and Automation (ICCCA)*, sivut 864–869, May 2017.

[Sim06] Simons, Matt: *Distributed Agile Development and the Death of Distance*. Cutter Consortium Sourcing and Vendor Relationships Advisory Service Executive Report, 5(4), 2006.

[Ste15] Stephen, Denning: *Agile: it's time to put it to use to manage business complexity.* 43(5):10–17, Jan 2015, ISSN 1087-8572. https://doi.org/10.1108/SL-07-2015-0057.

[Sub10] Subhas, Chandra Misra: *Identifying some critical changes required in adopting agile practices in traditional software development projects.* 27(4):451–474, Jan 2010, ISSN 0265-671X. https://doi.org/10.1108/02656711011035147.

[Tuo18] Tuomi, Jouni, kirjoittaja: *Laadullinen tutkimus ja sisällönanalyysi.* Kustannusosakeyhtiö Tammi, Helsinki, uudistettu laitos painos, 2018. https://www.ellibslibrary.com/hy/9789520400118.

[Ver] VersionOne, CollabNet: *13th annual state of agile report.* https://explore.versionone.com/state-of-agile/13th-annual-state-of-agile-report, vierailtu 16.10.2019 .

[Wan11] Wang, X.: *The Combination of Agile and Lean in Software Development: An Experience Report Analysis.* Teoksessa *2011 Agile Conference*, sivut 1–9, Aug 2011.